TESIS - KI142502

DETEKSI KLON SQL DI LAPISAN MODEL PADA APLIKASI BERARSITEKTUR MVC

FAWWAZ ALI AKBAR
5115201006

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom.,M.Eng

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

THESIS - KI142502

SQL CLONE DETECTION ON MODEL LAYER OF MVC-BASED APPLICATION

FAWWAZ ALI AKBAR
5115201006

SUPERVISOR
Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom.,M.Eng

MASTER PROGRAM
THE EXPERTISE OF SOFTWARE ENGINEERING
INFORMATICS ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

Oleh:
Fawwaz Ali Akbar
NRP. 5115201006

Dengan judul :
Deteksi Klon SQL di Lapisan Model pada Aplikasi Berarsitektur MVC

Tanggal Ujian : 17-7-2017
Periode Wisuda : 2016 Genap

Disetujui oleh:

Dr. Ir. Siti Rochimah, M.T
NIP. 196810021994032001 (Pembimbing 1)

Rizky Januar Akbar, S.Kom, M.Eng
NIP. 198701032014041001 (Pembimbing 2)

Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng.
NIP. 197411232006041001 (Penguji 1)

Fajar Baskoro, S.Kom, M.T
NIP. 197404031999031002 (Penguji 2)

Adhatus Sholichah, S.Kom, M.Sc
NIP. 198508262015042002 (Penguji 3)

Dekan Fakultas Teknologi Informasi,

Dr. Agus Zainal Arifin, S.Kom., M.Kom.
NIP. 197208091995121001

(halaman ini sengaja dikosongkan)

DETEKSI KLON SQL PADA LAPISAN MODEL DI APLIKASI BERARSITEKTUR MVC

Nama Mahasiswa **:** Fawwaz Ali Akbar
NRP **:** 5115201006
Pembibimbing **:** Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom.,M.Eng

ABSTRAK

Pola desain arsitektur *Model-View-Controller* (MVC) merupakan sebuah pola desain yang sesuai untuk sistem interaktif. Pola MVC banyak diadaptasi oleh aplikasi berbasis desktop maupun web. Banyak kerangka kerja dibangun dengan mengadaptasi pola MVC. Setiap lapisan pada MVC memiliki fungsi yang berbeda. Fungsi utama dari lapisan *Model* adalah pengaksesan data. Kerangka kerja berbasis MVC menyediakan kemudahan pengaksesan data seperti penulisan kueri yang dapat direpresentasikan dengan beberapa cara berbeda seperti *raw* SQL, *query builder*, ORM.

Dalam proses pengembangan sebuah sistem perangkat lunak, duplikasi kode merupakan masalah yang serius karena akan berdampak pada proses pemeliharaan sistem. Terkait dengan pola MVC dan kode klon, pendekatan deteksi klon yang ada saat ini masih belum efektif untuk mendeteksi klon di lapisan model. Hal ini dikarenakan sebagian besar kode di lapisan model adalah kueri ke sistem basis data. Sehingga seharusnya deteksi klon pada lapisan model juga mempertimbangkan duplikasi kueri. Duplikasi kueri dalam bentuk skrip SQL ini juga mempunyai dampak yang sama dengan dampak negatif kode klon. Misalnya, pada saat proses pemeliharaan sistem, ada sebuah kueri yang diubah, dan kueri tersebut sama dengan beberapa kueri yang lain, maka duplikat kueri yang lain juga memungkinkan untuk diubah. Sehingga informasi duplikat kueri sangat dibutuhkan. Penelitian ini mengusulkan metode deteksi duplikasi kueri pada lapisan model dengan mengidentifikasi duplikat SQL skrip yang dihasilkan dari *raw* SQL atau *query builder*. Hasil deteksi duplikasi kueri pada lapisan model ini dapat dijadikan acuan untuk menghindari dampak negatif dari kode klon.

Dari hasil uji coba pada pendekatan yang diusulkan menghasilkan akurasi sebesar 86,52%. Sedangkan untuk nilai precision dan recall berturut-turut sebesar 94,85% dan 84,34%.

Kata Kunci : *MVC, Web, SQL Klon, Lapisan Model, Deteksi Klon.*

(halaman ini sengaja dikosongkan)

SQL CLONE DETECTION ON MODEL LAYER OF MVC-BASED APPLICATION

Nama Mahasiswa : Fawwaz Ali Akbar
NRP : 5115201006
Pembibimbing : Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom.,M.Eng

ABSTRACT

Model-View-Controller (MVC) design pattern is design pattern that suitable for interactive systems. MVC is adapted in desktop and web-based applications. Moreover, many frameworks adapting MVC pattern. Each layer of MVC has a different function. The main function of the Model layer is data access. MVC-based framework providing ease of data access such as writing queries that can be represented in different ways such as raw SQL, query builder, ORM.

In software development, code duplication is a serious problem because it will impact on the maintenance process. Associated with the MVC pattern and code clone, clone detection approach that exists today is still not effective to detect clones in the model layer because most of the code in the model layer is the query to the database system. Based on that problem, clone detection of the model layer should considering duplicate query. Duplication queries also has same negative impact of code clones. For example, during system maintenance, if there is a query modified, and that query have duplicate, then the duplicate query is also possible to be changed. So that duplicate query information is needed.

This study tries to detect duplicate queries in the model layer by identifying duplicate SQL scripts generated from the raw SQL or query builder. Detected duplicate query in model layer can be used as a reference in order to avoid the negative impact of the code clones. The results of evaluation on the proposed approach yield accuracy 86.52%, precision 94.85% and 84.34% recall.

Keywords : *MVC, Web , SQL Clone, Model layer, Clone Detection*

(halaman ini sengaja dikosongkan)

KATA PENGANTAR

Segala puji syukur kepada Allah SWT yang telah melimpahkan rahmat dan hidayah Nya sehingga penulis dapat menyelesaikan Tesis yang berjudul " Deteksi Klon SQL pada Lapisan Model di Aplikasi Berarsitektur MVC" sesuai dengan target dan waktu yang diharapkan.

Proses pembuatan dan pengerjaan Tesis ini merupakan pengalaman yang sangat berharga bagi penulis untuk memperdalam ilmu pengetahuannya khususnya di bidang rekayasa perangkat lunak dan teknologi informasi. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada :

1. Allah SWT atas limpahan rahmat Nya sehingga penulis dapat menyelesaikan Tesis ini dengan baik.
2. Ibu Dra. Siti Zulfah dan Bapak Drs. Sanuddin selaku orang tua penulis yang selalu mendoakan penulis agar senantiasa diberi kelancaran dalam menyelesaikan tesis ini.
3. Ibu Dr. Ir. Siti Rochimah, M.T. dan Bapak Rizky Januar Akbar, S.Kom.,M.Eng. selaku dosen pembimbing penulis yang telah memberikan kepercayaan, perhatian, bimbingan, bantuan dan motivasi kepada penulis dalam proses menyelesaikan tesis ini.
4. Bapak Daniel Oranova Siahaan S.Kom, M.Sc, PD.Eng., Ibu Adhatus Sholichah S.Kom., M.Sc. dan Bapak Fajar Baskoro S.Kom, M.T. selaku Dosen Penguji yang telah memberikan bimbingan, arahan, nasehat dan koreksi dalam pengerjaan tesis ini.
5. Bapak Waskitho Wibisono, S.Kom., M.Eng., PhD. selaku ketua program Pascasarjana Teknik Informatika ITS serta Dosen Pascasarjana Teknik Informatika ITS lainnya yang telah memberikan ilmunya.
6. Mbak Lina, Mas Kunto dan segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
7. Saudara penulis M. Hilman Fuadil Amin, Nur Indradewi O, Miqdad Agung Z, Hanny, Hilda Izzati M, Agus Tamam M, Akhmad Cahyo K, Kamalatul

K. serta seluruh keluarga besar penulis yang selalui memberikan dukungan dan semangat kepada penulis.

8. Keponakan penulis Muhammad Rafif Nur ‘And, Salman Nazran El-Syarif, Kaniya, Radinna, Dzawai yang selalu memberikan keceriaan dan semangat kepada penulis.
9. Teman-teman seperjuangan Dika Rizky, Septiyawan Rosetya W, M. Sonhaji, Wawan Gunawan, Andreyan, Nur Fajri Azhar, Didih serta teman-teman angkatan 2015 lain yang selalu ada di saat penulis mengalami suka dan duka.
10. Tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu disini yang telah membantu terselesaikannya Tesis ini

Penulis menyadari bahwa Tesis ini masih jauh dari kesempurnaan dan banyak kekurangan. Untuk itu dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari para pembaca.

Surabaya, Juli 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

(halaman ini sengaja dikosongkan)

DAFTAR TABEL

BAB 1
PENDAHULUAN

Pada bab ini akan dijelaskan mengenai beberapa hal dasar dalam penelitian yang meliputi latar belakang, perumusan masalah, tujuan, manfaat, kontribusi penelitian, dan batasan masalah.

1.1. Latar Belakang

Pola desain arsitektur *Model-View-Controller* (MVC) merupakan sebuah pola desain yang sesuai untuk sistem interaktif (Leff & Rayfield, 2001). Pola desain MVC membagi sebuah sistem menjadi tiga bagian atau lapisan yaitu *Model*, *View*, dan *Controller*. Lapisan *Model* berhubungan dengan data, Lapisan *View* berhubungan dengan presentasi dan lapisan *Controller* berhubungan dengan interaksi pengguna (Buschmann, Henney, & Schmidt, 2007).

Pola MVC banyak diadaptasi oleh aplikasi berbasis desktop maupun web. Selain itu, banyak kerangka kerja dibagun dengan mengadaptasi pola MVC sebagai pola desain arsitekturnya. Kerangka kerja berbasis MVC yang ada saat ini tidak hanya memisahkan antara presentasi, data, dan logika saja. Tetapi menyediakan fungsionalitas lain seperti kemudahan pengaksesan data. Pengaksesan data terletak pada lapisan *Model*. Bentuk pengaksesan data pada lapisan model yang paling umum adalah kueri ke sistem basis data. Pada kerangka kerja MVC, penulisan kueri tersebut dapat direpresentasikan dengan beberapa cara yang berbeda seperti *raw* SQL, *query builder*, ORM (Jound, 2016).

Dalam proses pengembangan sebuah sistem perangkat lunak, duplikasi kode atau kode klon merupakan masalah yang serius karena akan berdampak pada proses pemeliharaan sistem seperti meningkatnya biaya pemeliharaan sistem dan perambatan *bug* (Rattan, Bhatia, & Singh, 2013). Menurut penelitian sebelumnya, keberadaan kode klon dalam sistem cukup besar. Kode klon terjadi sekitar 20-30% dalam sebuah sistem yang besar (Rattan et al., 2013), sedangkan pada aplikasi berbasis web ditemukan 17-63% klon (Rajapakse & Jarzabek, 2005).

Terkait dengan pola MVC dan kode klon, pendekatan deteksi klon yang ada saat ini masih bersifat umum untuk keseluruhan kode pada suatu proyek perangkat lunak. Padahal, pada MVC, setiap lapisan memiliki fungsi dan tujuan yang berbeda. Pada lapisan model yang berfokus pada pengaksesan data, proses deteksi klon konvensional masih belum mencukupi. Hal ini dikarenakan sebagian besar kode di lapisan model adalah kueri ke sistem basis data. Sehingga seharusnya deteksi klon pada lapisan model juga mempertimbangkan duplikasi kueri untuk pengaksesan data ke sistem basis data. Selain itu, dengan adanya beberapa opsi penulisan kueri seperti *raw* SQL, *query builder*, dan ORM, yang ditawarkan oleh kerangka kerja berbasis MVC memungkinkan duplikat kueri dengan bentuk berbeda. Pada proyek skala besar yang memiliki lebih dari satu programmer, kemungkinan terjadinya duplikat kueri tersebut semakin besar karena adanya beberapa opsi penulisan kueri sehingga suatu kueri memiliki struktur kode yang berbeda tetapi memiliki fungsionalitas kueri yang sama. Pada level kode kerangka kerja, duplikasi tersebut lebih sulit dideteksi karena kueri tersebut memiliki struktur yang berbeda.

Kesamaan fungsionalitas dari kueri tersebut dapat dilihat dari kesamaan skrip SQL yang dihasilkan dari beberapa opsi penulisan kueri seperti *raw* SQL, *query builder*, dan ORM, yang ditawarkan oleh kerangka kerja berbasis MVC. Duplikasi kueri atau skrip SQL ini juga mempunyai dampak yang sama dengan dampak negatif kode klon. Misalnya, pada saat proses pemerliharaan sistem, ada sebuah kueri yang diubah, dan kueri tersebut sama dengan beberapa kueri yang lain, maka duplikat kueri yang lain juga memungkinkan untuk diubah. Sehingga informasi duplikat kueri sangat dibutuhkan. Keuntungan lain yang bisa didapatkan adalah hasil deteksi klon pada skrip SQL dapat digunakan untuk rekomendasi pembuatan pustaka untuk lapisan model.

Muhammad (Muhammad, Zibran, Yamamoto, & Roy, 2013) melakukan studi kode klon pada web tradisional dan web berbasis kerangka kerja MVC. Hasilnya, kode klon pada web tradisional lebih sedikit dibandingkan web berbasis kerangka kerja MVC. Hal tersebut dikarenakan dalam kerangka kerja sudah diatur bagaimana menggunakan kode (template) pada kerangka kerja MVC. Berasarkan persebaran lokasi kode klon, densitas (kerapatan) kode klon lebih tinggi pada web

berbasis kerangka kerja MVC. Artinya, kode klon yang terdeteksi terlokalisir pada modul-modul tertentu (40%). Sedangkan kode klon pada web struktural lebih menyebar yaitu 70% dari keseluruhan file. Selain itu, pada kerangka kerja berbasis MVC tidak ditemukan klon antar lapisan pada lapisan *model-view-controller*. Islam (Islam, Islam, Islam, & Halim, 2011) juga melakukan penelitian yang hampir sama. Islam dkk. melakukan studi perbandingan kode klon pada kerangka kerja untuk web berbasis MVC dan bukan MVC. Mereka menyatakan bahwa kerangka kerja yang tidak menggunakan pola MVC lebih rentan terjadi kode klon dibandingkan dengan kerangka kerja MVC. Selain itu, mereka menyatakan hal yang sama bahwa persebaran kode klon lebih besar pada kerangka kerja bukan MVC. Muhammad dkk. dan Islam dkk. tidak secara khusus mengidentifikasi duplikasi kueri pada lapisan model. Identifikasi duplikasi kueri pada lapisan model merupakan hal yang penting, karena fungsi utama dari lapisan model adalah melakukan akses data, sehingga yang perlu diidentifikasi bukan hanya pada struktur kode tetapi fungsionalitas dari kode yang direpresentasikan dalam bentuk kueri.

Selain itu, beberapa penelitian terkait kode klon telah dilakukan pada aplikasi berbasis web. Aplikasi berbasis web merupakan aplikasi yang dibangun dengan beberapa bahasa pemrograman sekaligus. Penelitian deteksi duplikasi kode pada bahasa pemrograman yang dipakai pada web, misalnya HTML (Cordy & Dean, 2004), Javascript (Cheung, Ryu, & Kim, 2016) dan CSS (Mazinanian, Tsantalis, & Mesbah, 2014), pernah dilakukan. Penelitian pada HTML, Javascript dan CSS tersebut dapat dikaitkan dengan lapisan *view* pada MVC. Sedangkan pada lapisan *controller* yang biasanya dibangun dengan bahasa pemrograman seperti PHP, Java, dan bahasa scripting lain. Bahasa pemrograman tersebut telah diakomodasi oleh kakas bantu yang sudah ada, misalnya PHP dan Java dapat menggunakan kakas bantu hasil dari penelitian Jian (Jiang, Misherghi, & Su, 2007). Tetapi penelitan yang secara spesifik membahas duplikasi SQL masih sangat sedikit, khususnya di lapisan *model* pada MVC. Padahal SQL sangat erat kaitannya dengan aplikasi berbasis web.

Berdasarkan masalah duplikasi kueri pada lapisan model yang telah dijelaskan sebelumnya. Penelitian ini mencoba untuk melakukan deteksi duplikasi

kueri pada lapisan model dengan mengidentifikasi duplikat SQL skrip yang dihasilkan dari *raw* SQL atau *query builder*. Kueri yang direpresentasikan dalam bentuk *raw* SQL, *query builder*, dan ORM akan ditranslasikan kedalam bentuk skrip SQL. Pada level skrip SQL pun masih memungkinkan adanya duplikasi yang disebabkan variasi skrip SQL misalnya penggunaan nama alias tabel dan kolom, struktur urutan kolom dan tabel. Hasil deteksi duplikasi kueri pada lapisan model ini dapat dijadikan acuan untuk menghindari dampak negatif dari kode klon dan sebagai rekomendasi pembuatan pustaka untuk lapisan model.

1.2. Perumusan Masalah

Berdasarkan latar belakang penelitian ini maka rumusan masalah dapat dijabarkan menjadi beberapa butir berikut ini.

1. Bagaimana mendefinisikan tipe klon pada skrip SQL?
2. Bagaimana melakukan deteksi duplikasi skrip SQL pada lapisan model?
3. Bagaimana mengevaluasi hasil deteksi duplikasi skrip SQL?

1.3. Tujuan

Tujuan dari tesis ini adalah mengusulkan pendekatan untuk deteksi duplikasi kueri pada lapisan model. Diharapkan dengan pendekatan ini dapat mengidentifikasi duplikasi kueri pada lapisan model yang memiliki struktur yang berbeda tetapi memiliki fungionalitas sama.

1.4. Manfaat

Manfaat dalam penelitian ini terkait dengan deteksi klon SQL pada lapisan model adalah sebagai berikut:

1. Penelitian ini diharapkan dapat mendeteksi klon SQL di lapisan model pada lapisan MVC.
2. Hasil deteksi tersebut dapat digunakan sebagai salah satu acuan pada proses pemeliharaan dan pengembangan sistem. Seperti, menggunakan informasi hasil deteksi untuk menghilangkan duplikat klon dengan cara membuat pustaka untuk lapisan model. Sehingga dapat mengurangi kompleksitas dan ukuran kode sistem.

1.5. Kontribusi Penelitian

Kontribusi dalam penelitian ini terkait dengan deteksi klon SQL pada lapisan model adalah sebagai berikut:

1. Pendefinisian tipe klon pada skrip SQL.
2. Membuat pendekatan baru untuk deteksi klon SQL pada lapisan model.

1.6. Batasan Masalah

Batasan dalam penilitian ini adalah sebagai berikut.

1. Penelitian ini hanya berfokus pada sistem berbasis web yang menggunakan kerangka kerja berbasis MVC CodeIgniter.
2. Bahasa pemrograman yang dijadikan obyek penelitian adalah PHP.
3. Penelitian ini hanya berfokus di DML (Data Manipulation Language) SQL dan dikhususkan pada SELECT *statement*.
4. Bentuk query pada layer model yang digunakan adalah *raw* SQL dan *query builder* yang disediakan kerangka kerja.
5. Pada penelitian ini deteksi SQL klon hanya berdasarkan pada teks kode.
6. Standar SQL yang digunakan pada penelitian ini mengacu pada standar yang diakomodasi oleh MySQL versi 5.1.
7. Penelitian ini hanya berfokus pada deteksi SQL klon. Sehingga hasil dari penelitian ini adalah informasi SQL klon di lapisan model pada MVC.
8. Pada penelitian ini proses ekstraksi kueri dilakukan dengan pendekatan statis. Pendekatan statis adalah pendekatan tanpa menjalankan kode program. Selain itu, pada penelitian ini mengabaikan adanya persyaratan dari alur kontrol.

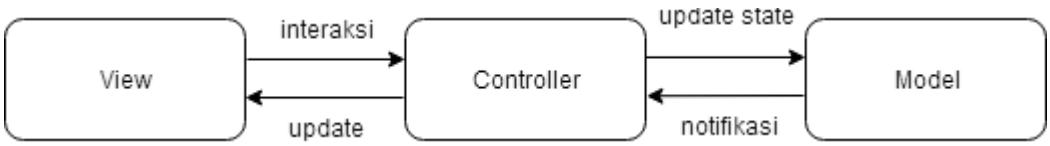
(halaman ini sengaja dikosongkan)

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

Pada Bab 2 ini dijelaskan konsep dasar tentang teori dan kajian pustaka yang digunakan sebagai landasan dalam melakukan penelitian.

2.1. Arsitektur *Model-View-Controller* (MVC)

Pola desain arsitektur *Model-View-Controller* (MVC) merupakan sebuah pola desain yang sesuai untuk sistem interaktif (Leff & Rayfield, 2001). Pola desain MVC membagi sebuah sistem menjadi tiga bagian atau lapisan yaitu *Model*, *View*, dan *Controller*. Lapisan *Model* berhubungan dengan data, Lapisan *View* berhubungan dengan presentasi dan lapisan *Controller* berhubungan dengan interaksi pengguna (Buschmann et al., 2007). Gambar 2.1. berikut ini merupakan gambaran umum dari interaksi lapisan *model, view* dan *controller*. Pada gambar tersebut menunjukkan bahwa lapisan *view* berinteraksi dengan lapisan *controller*, kemudain lapisan *controller* merubah *state* dari lapisan *model*. Perubahan *state* lapisan *model* akan diteruskan ke *view* lewat lapisan *controller*.



Gambar 2.1. Pola interaksi Model-View-Controller

Perkembangan dan pengadaptasian pola MVC pada banyak sistem (desktop dan web) menjadikan pola MVC memiliki banyak variasi pola interaksi dan komponen pada lapisan *Model*, *View* dan *Controller* (Morales-Chaparro, Linaje, Preciado, & Sánchez-Figueroa, 2007).

Banyak kerangka kerja dibangun dengan mengadaptasi pola MVC sebagai pola desain arsitekturnya. Pada web dan aplikasi berbasis web, adaptasi pola MVC pada sebuah kerangka kerja begitu populer. Selain itu, kerangka kerja berbasis MVC yang ada saat ini tidak hanya memisahkan antara presentasi, data, dan logika saja. Tetapi menyediakan fungsionalitas lain seperti kemudahan pengaksesan data.

Pengaksesan data terletak pada lapisan *Model*. Bentuk pengaksesan data pada lapisan model yang paling umum adalah kueri ke sistem basis data. Pada kerangka kerja MVC, penulisan kueri tersebut dapat direpresentasikan dengan beberapa cara yang berbeda seperti *raw* SQL, *query builder*, ORM (Jound, 2016). Pada penelitian ini yang digunakan hanya *raw* SQL dan *query builder.* Berikut penjelasan *raw* SQL dan *query builder* (Jound,2016) :

- ***Raw* SQL** : proses penulisan query langsung ditulis dengan bahasa SQL. Contoh penggunaan *raw* SQL pada kerangka kerja MVC sebagai berikut:

```
$query = $this->db->query('select nrp, nama from mahasiswa');
```

- ***Query Builder** :* merupakan salah satu fitur dalam kerangka kerja web MVC yang merepresentasikan kueri basis data ke dalam bentuk kode pemrograman. Contoh penggunaan *query builder* pada kerangka kerja MVC sebagai berikut:

```
$this->db->select('nrp, nama');
$this->db->from('mahasiswa');
$query = $this->db->get();
```

2.2. Kode Klon

Kloning perangkat lunak adalah kegiatan menyalin dan menempelkan fragmen kode yang sudah ada, dengan atau tanpa modifikasi, ke dalam kode lain pada proses pengembangan perangkat lunak. Hasil dari kegiatan menyalin dan menempelkan fragmen kode adalah klon perangkat lunak, atau lebih sering dikenal sebagai kode klon (Rattan et al., 2013).

Ada beberapa pro dan kontra tentang keberadaan kode klon dalam sebuah sistem. Pro dan kontra tersebut dilihat dari keuntungan dan kerugian yang diakibatkan kode klon dalam suatu sistem. Salah satu keuntungan kode klon adalah mempercepat proses pengembangan perangkat lunak jika terjadi perubahan kebutuhan perangkat lunak. Sedangkan kerugian kode klone yang paling sering dibahas adalah meningkatnya biaya perawatan sistem dan perambatan *bug* (Rattan

et al., 2013). Alasan pengembang melakukan kode klon adalah karena keterbatasan waktu dan sumber daya yang ada (Rattan et al., 2013).

Selain itu, keberadaan kode klon dalam sistem cukup besar. Beberapa penelitian menyatakan bahwa kode klon terjadi sekitar 20-30% dalam sebuah sistem yang besar (Rattan et al., 2013), sedangkan pada web ditemukan 17-63% klon (Rajapakse & Jarzabek, 2005).

Kode klon secara umum dibagi menjadi 4 definisi atau tipe, yaitu tipe 1 (*exact clone*), 2 (*renamed clone*), 3 (*near-miss clone*) dan 4 (*semantic clone*). Berikut ini penjelasan tipe klon tersebut (Rattan et al., 2013):

- Tipe 1 (*exact clone*) : Sepasang fragmen kode memiliki tulisan yang sama persis kecuali adanya penambahan spasi dan komentar.
- Tipe 2 (*renamed clone*) : sepasang fragmen kode yang memiliki struktur yang sama, perubahan ada pada penamaan (identifier), literal, tipe data, layout dan komentar.
- Tipe 3 (*near-miss clone*) : Klon tipe 2 disertai penambahan atau penghapusan fragment code.
- Tipe 4 (*semantic clone*) : sepasang fragmen kode dinyatakan klon Tipe 4 jika memiliki fungsionalitas yang sama tetapi tidak memiliki kesamaan struktur atau text.

Definisi tipe klon tersebut masih belum baku sepenuhnya. Sebuah survey yang dilakukan oleh Chatterji (Chatterji, Carver, & Kraft, 2012) menyatakan bahwa para peneliti dibidang kode klon masih belum sepakat dan sama mengenai definisi klon. Dalam temuan survey tersebut menyatakan bahwa sebagian besar peneliti sepakat untuk definisi kode klon tipe 1 dan 2, sedangkan untuk tipe 3 dan 4 banyak perbedaan pendapat di kalangan peneliti.

2.2.1. Kode Klon di Web MVC

Dalam pengembangan web, duplikasi kode atau kode klon merupakan masalah yang serius karena akan berdampak pada proses pemeliharaan sistem. Menurut penelitian sebelumnya, kode klon pada web masih tinggi yaitu 17% hingga 63% kode klon yang terdeteksi (Rajapakse & Jarzabek, 2005).

Muhammad dkk. (Muhammad et al., 2013) melakukan studi kode klon pada web tradisional dan web berbasis kerangka kerja MVC. Hasilnya, kode klon pada web tradisional lebih sedikit dibandingkan web berbasis kerangka kerja MVC. Hal tersebut dikarenakan dalam kerangka kerja sudah diatur bagaimana menggunakan kode (template) pada kerangka kerja MVC. Berasarkan persebaran lokasi kode klon, densitas (kerapatan) kode klon lebih tinggi pada web berbasis kerangka kerja MVC. Artinya, kode klon yang terdeteksi terlokalisir pada modul-modul tertentu (40%). Sedangkan kode klon pada web struktural lebih menyebar yaitu 70% dari keseluruhan file. Selain itu, pada kerangka kerja berbasis MVC tidak ditemukan klon antar lapisan pada lapisan model-view-controller. Islam dkk. (Islam et al., 2011) juga melakukan penelitian yang hampir sama. Islam dkk. melakukan studi perbandingan kode klon pada kerangka kerja untuk web berbasis MVC dan bukan MVC. Mereka menyatakan bahwa kerangka kerja yang tidak menggunakan pola MVC lebih rentan terjadi kode klon dibandingkan dengan kerangka kerja MVC. Selain itu, mereka menyatakan hal yang sama bahwa persebaran kode klon lebih besar pada kerangka kerja bukan MVC.

Seperti dijelaskan pada poin sebelumnya bahwa kerangka kerja web MVC saat ini mengakomodasi beberapa cara pengaksesan ke basis data. Salah satunya adalah *raw* SQL dan *sql builder*. Pada proyek skala besar yang memiliki lebih dari satu programmer, ada kemungkinan terjadi duplikasi kueri pada lapisan model. Selain itu, dengan adanya beberapa opsi penulisan kueri seperti *raw* SQL, *query builder*, dan ORM, memungkinkan duplikat kueri dengan bentuk berbeda. Memiliki struktur kode yang berbeda tetapi memiliki fungsionalitas kueri yang sama. Pada level kode kerangka kerja, duplikasi tersebut lebih sulit dideteksi karena kueri tersebut memiliki struktur yang berbeda. Contoh kasus penggunaan struktur kueri yang berbeda (*raw* SQL dan *query builder* pada kerangka kerja MVC Codeigniter) tetapi memiliki fungsionalitas kueri yang sama:

Tabel 2.1. Contoh kasus *raw* SQL dan *query builder*

Query builder	*Raw* SQL	SQL skrip
`$query = $this->db->get('tabel1');`	`$query = 'select * from tabel1';` `$query = $this->query($query)`	`select * from tabel1`

Query builder	*Raw* SQL	SQL skrip

`$this->db->select('nrp, nama, jurusan');`
`$this->db->from('mahasiswa');`
`$query = $this->db->get();`
	`$query = $this->query(‘select nrp, nama, jurusan from mahasiswa’);`	`select nrp, nama, jurusan from mahasiswa`

`$this->db->select('nrp, nama, jurusan');`
`$this->db->from('mahasiswa');`
`$this->db->where(‘jurusan’,’informatika’);`
`$query = $this->db->get();`
	`$query = ‘select nrp, nama, jurusan from mahasiswa where jurusan = informatika';`
	`$query = $this->query($query)`
		`select nrp, nama, jurusan from mahasiswa where jurusan = informatika`

`$this->db->select('nrp, nama, jurusan, fakultas');`
`$this->db->from(mahasiswa);`
`$this->db->join('fakultas', 'mahasiswa.id_fakultas = fakultas.id_fakultas');`
`$query = $this->db->get();`
	`$query = $this->query(‘select nrp, nama, jurusan, fakultas from mahasiswa join fakultas on mahasiswa.id_fakultas = fakultas.id_fakultas’);`	`select nrp, nama, jurusan, fakultas from mahasiswa join fakultas on mahasiswa.id_fakultas = fakultas.id_fakultas`

2.3. SQL (*Structured Query Language*)

Hampir semua aplikasi membutuhkan tempat penyimpanan data. Tempat penyimpanan data yang paling umum digunakan adalah sistem basis data. Interaksi sistem dengan basis data yang paling umum adalah menggunakan kueri. Bahasa yang digunakan untuk kueri data seperti memasukkan data, mengubah data dan mengambil data adalah bahasa SQL atau *Structured Query Language*. *Statement* bahasa SQL dibagi menjadi kategori yang memiliki fungsi berbeda. Berikut penjelasan terkait kategori SQL *statement*.

2.3.1. Kategori SQL *Statement*

SQL *statement* dikemompokkan menjadi tiga kelompok utama yaitu SQL *schema statement*, SQL *data statement*, dan SQL *transaction statement* (Beaulieu, 2009).

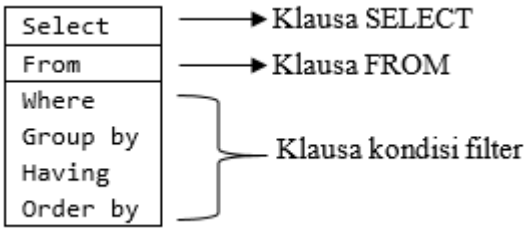
- SQL *schema statement:* SQL *statement* yang memiliki fungsi mendefinsiskan struktur data pada sistem basis data.
- SQL *data statement:* SQL *statement* yang memiliki fungsi untuk memanipulasi struktur data yang telah didefinisikan oleh SQL *schema statement*.

- SQL *transaction statement:* SQL *statement* yang memiliki fungsi utama untuk menangani sebuah transaksi. *Statement* ini menggunakan kalusa *begin, end* dan *rollback.*

Pada penelitian ini hanya berfokus pada kelompok SQL *data statement.* Dan difokuskan lagi pada pengambilan data atau *statement* SELECT. Pada *statement* SELECT terdiri dari tiga klausa yang berbeda yaitu select, from dan where.

```
SELECT kolom, kolom,..
FROM tebel,…
WHERE kondisi,…
```

Pada klausa SELECT digunakan untuk menentukan kolom yang akan ditampilkan pada sebuah kueri. Pada klausa FROM digunakan mendefinisikan tabel-tabel yang terlibat pada sebuah kueri. Terakhir adalah kalusa WHERE yang digunakan untuk mendefinisikan kondisi untuk kueri yang dibuat. Selain WHERE ada beberapa klausa yang dapat digunakan untuk tujuan spesifik seperti GROUP BY, HAVING, dan ORDER BY (Beaulieu, 2009). Pada penelitian ini sebuah *statement* SELECT akan dibagi menjadi beberapa bagian, yaitu SELCET, FROM, dan klausa filter. Klausa filter meliputi WHERE, GROUP BY, HAVING, dan ORDER BY.



Gambar 2.2. Bagian SELECT *statement* pada penelitian

2.3.2. Standar SQL dan *Database Management System* (DBMS)

Bahasa SQL memiliki standar dan versi yang berbeda-beda. Ada dua standar pada SQL yaitu ANSI dan ISO. Tetapi kedua standar tersebut identik (Oracle, 2017).

Database Management System (DBMS) merupakan sistem yang mengelola basis data. Sistem tersebut menggunakan standar SQL untuk mengelola basis data. Saat ini banyak vendor yang menyediakan DBMS seperti ORACLE, MySQL, DB2,

MariaDB, Postgre, dan sebagainya. Setiap vendor atau produk DBMS mengimplementasikan standar SQL dengan cara yang berbeda-beda (Arvin, 2014). Pada beberapa produk misalnya MySQL (MySQL, 2017) menyatakan bahwa mereka tidak mengimplementasikan standar SQL sama persis, mereka melakukan pengembangan dari standar SQL yang ada.

Berdasarkan pada perbedaan implementasi standar SQL dari berbagai vendor atau produk DBMS, maka pada penelitian ini mengacu langsung pada salah satu vendor yaitu MySQL. Alasan dipilihnya MySQL karena MySQL merupakan DBMS yang paling umum dan popular digunakan, khususnya pada domain penelitian ini yaitu di area aplikasi berbasis web.

Selain itu, untuk versi MySQL yang digunakan adalah versi 5.1. versi tersebut mengacu pada kebutuhan server kerangka kerja MVC CodeIgniter yang digunakan pada penelitian ini.

2.3.3. SQL Query Pattren

Anderson dan Hills melakukan penelitian pada web untuk mengetahui jenis konstruksi pola kueri yang sering dilakukan oleh programer (Anderson & Hills, n.d.). Pola-pola kueri pada pengembangan web dapat dimanfaatkan dalam penelitian ini. Pola-pola kueri tersebut akan digunakan untuk mengekstraksi kemungkinan kueri pada aplikasi web. Karena pada penelitian ini berfokus pada lapisan model pada web MVC, dimana model merupakan suatu layer berisi kueri-kueri ke basis data. Berikut ini merupakan pola-pola konstruksi kueri pada web (Anderson & Hills, 2017.) :

1. **String Kueri Literal *(Literal Query Strings) – QCP1***

   Pola pertama adalah menggunakan string literal. Pola ini ada dua jenis yaitu pola eksekusi langsung dalam bentuk string kueri atau disimpan pada variable kemudian dieksekusi. Contoh:

   ```
   $query = mysql_query("SELECT termid, title FROM terms");
   ```

   Atau

   ```
   $q = "SELECT id FROM Faq WHERE parent_id = 0 ORDER BY id";
   $result0 = mysql_query ($q, $dbLink);
   ```

2. **Penggabungan Variabel *(Cascading Concatenating Assignments) – QCP2***

   Pola kedua adalah kueri yang dikonstruksi dari penggabungan variable. Penggabungan variabel dengan menggunakan operator (.=) bukan dengan (=). Contoh dari penggunaan penggabungan variable sebagai berikut:

```
$q = "SELECT * FROM Faq WHERE parent_id = 0 AND publish = 'y'
";
$q .= "ORDER BY list_order"; $result0 = mysql_query ($q,
$dbLink);
```

3. **Variable Terdistribusi Pada Alur Kontrol (*Assignments Distributed over Control Flow*) - *QCP3***

   Pola ketiga adalah kueri yang dikonstruksi dari satu variabel dimana variabel tersebut tersebar pada alur percabangan. Dan pada setiap alur percabangan variable tersebut memungkinkan memiliki string kueri yang berbeda. Contoh :

```
$sql = "SELECT * FROM resources WHERE type='encyclopedia'
ORDER BY name";
if($order_by == "votes"){
      $sql="SELECT * FROM resources WHERE type='encyclopedia'
      ORDER BY votes DESC";
} elseif ($order_by == "average"){
      $sql="SELECT * FROM resources WHERE type='encyclopedia'
      ORDER BY average DESC";
}
$result = mysql_query($sql, $db);
```

   Pola ketiga ini memiliki dua sub-pola yaitu:

   - QCP3a: penugasan literal kueri pada satu variabel yang terdistribusi pada alur kontrol. Seperti pada contoh diatas.
   - QCP3b: pola QCP4 yang terdistribusi pada alur control.

4. **Kueri String Dinamis (*Dynamic Query Strings*) – QCP4**

   Pola keempat adalah pola kueri yang mengandung parameter. Pada pola keempat ini dibagi menjadi tiga sub-pola, yaitu:

   - QCP4a: penyisipan literal fragmen, variabel, pemanggilan fungsi dan kemungkinan penyisipan non-literal lain pada kueri string.
   - QCP4b: operasi rangkaian antara kueri string dengan literal fragmen, variabel, pemanggilan fungsi.
   - QCP4c: variabel yang berisi pola QCP4a atau QCP4b.

   Contoh:

```
$query = mysql_query("SELECT title FROM semesters WHERE semesterid = $_POST[semester]");
```
[QCP4a]

```
$q = "SELECT * FROM Faq WHERE parent_id = $id";
$result = mysql_query ($q, $dbLink);
```
[QCP4b]

2.3.4. Klon SQL

Penelitian dibidang deteksi kode klon telah banyak dilakukan dan mencakup banyak bahasa pemrograman seperti Java, C, php, dan sebagainya. Deteksi klon juga dilakukan pada level yang lebih awal (fase desain) pada fase pengembangan perangkat lunak. Deteksi klon pada fase desain tersebut disebut model klon.

Deteksi klon juga dilakukan pada aplikasi web. Aplikasi berbasis web merupakan aplikasi yang biasanya dibangun dengan beberapa bahasa pemrograman sekaligus. Penelitian deteksi duplikasi kode pada bahasa pemrograman yang dipakai pada web, misalnya HTML (Cordy & Dean, 2004) dan CSS (Mazinanian et al., 2014), pernah dilakukan. Tetapi penelitan yang secara spesifik membahas duplikasi SQL masih sangat sedikit. Padahal SQL sangat erat kaitannya dengan aplikasi web.

Berikut ini merupakan beberapa penelitian terkait dengan deteksi klon pada area web dan MVC.

Tabel 2.2. Penelitian Sebelumnya

Peneliti	**Objek**	**Penelitian**
Muhammad, Zibran, Yamamoto, & Roy, 2013	Web MVC .NET	Melakukan studi kode klon pada web tradisional dan web berbasis kerangka kerja MVC. Tidak secara khusus membahas klon setiap lapisan MVC dan tidak memberikan penanganan secara khusus setiap lapisan.
Islam, Islam, Islam, & Halim, 2011	Web MVC .NET	Melakukan penelitian yang hampir sama dengan Muhammad dkk. Pada penelitian Islam dkk. melakukan studi perbandingan kode klon pada kerangka kerja untuk web berbasis MVC dan bukan MVC. Tidak secara khusus membahas klon setiap lapisan dan tidak memberikan penanganan secara khusus setiap lapisan.
Cordy & Dean, 2004	HTML	Melakukan penelitian deteksi klon pada kode HTML. Sebelum melakukan proses deteksi, terlebih dahulu diekstraksi kode-kode HTML pada kode sumber. Karena web dibangun dengan beberapa bahasa pemrorgaman. Terkait dengan MVC, deteksi ini dapat digunakan untuk lapisan *view*.
Mazinanian et al., 2014	CSS	Mazinanian dkk, tidak secara spesifik melakukan deteksi klon pada CSS. Fokus penelitiannya adalah refactoring dokumen CSS. Dan fokus refactoring CSS adalah menghilangkan duplikat kode CSS. Sehingga sebelum melakukan refactoring, terlebih dahulu dilakukan deteksi klon pada kode CSS.

Peneliti	Objek	Penelitian
Cheung, Ryu, & Kim, 2016	Javascript	Melakukan penelitian dibidang deteksi klon dengan objek penelitian di Javascript. Terkait dengan MVC, deteksi ini dapat digunakan untuk lapisan view untuk penggunaan Javascript pada area lapisan *view*.
Dobarkod, 2017	SQL	Melakukan deteksi duplikasi SQL pada framework Django. Model deteksi yang digunakan masih sederhana, hanya mengakomodasi duplikat SQL yang memiliki struktur sama persis (*exact clone*). Selain itu, deteksi duplikat kueri ini hanya melakukan komparasi kueri pada setiap HTTP Request, sehingga hanya melakukan deteksi pada kueri-kueri yang dieksekusi pada setiap HTTP Request. Tidak komprehensif pada keseluruhan kode. Tetapi, deteksi duplikat kueri pada Django tersebut merupakan bukti bahwa duplikasi SQL perlu dilakukan. Perbedaan dengan penelitian yang dilakukan adalah lebih komprehensif karena proses deteksi dilakukan pada level kode dilapisan model. Selain itu didefinisikan tipe-tipe klon SQL.

Bahasa SQL berbeda dengan bahasa pemrograman konvensional sehingga untuk mendeteksi klon pada SQL terlebih dahulu akan didefinisikan tipe-tipe klon pada SQL. Tipe-tipe klon pada SQL ini merupakan salah satu kontribusi pada penelitian ini.

2.4. Deteksi Klon

Proses deteksi klon pada kode aplikasi atau sistem memiliki perkembangan pesat. Banyak penelitian yang membahas proses, pendekatan, dan teknik pendeteksian klon pada kode aplikasi atau sistem.

2.4.1. Jenis Deteksi Kode Klon

Beberapa penelitian menghasilkan beberapa tools yang merepresentasikan pendekatan dan teknik tertentu (Rattan et al., 2013). Sejumlah pendekatan dan teknik tersebut dapat dikelompokkan sebagai berikut (Rattan & Kaur, 2016) :

- **Teknik Berbasis Text** : Pendekatan teknik ini adalah melakukan komparasi kode setiap baris kode dengan direpresentasian dengan string tanpa ada transformasi kode. Tetapi, saat ini perkembangan deteksi klon berbasis text sudah mampu mendeteksi hingga klon tipe 3 (Rattan & Kaur, 2016) (Cordy & Roy, 2011).
- **Teknik Berbasis Token** : pada pendekatan ini, kode akan diubah menjadi serangkaian token. Pada rangkaian token ini akan dicari kemiripan dari rangkain token dengan fragmen kode yang lain. Sebelum proses komparasi kode akan melawati proses tokenisasi. Teknik berbasis token ini memiliki performa lebih rendah dibandingkan dengan teknik berbasis text karena adanya proses tokenisasi. Pendekatan berbasis token ini bisa mendeteksi kode klon tipe 1,2, dan 3. Kakas bantu deteksi klon berbasis token yang popular adalah CCFinder (Rattan & Kaur, 2016) (Kamiya, Kusumoto, & Inoue, 2002).
- **Teknik Berbasis *Abstract Syntax Tree* (AST) :** pada pendekatan ini, kode akan ditransformasikan menjadi *Abstract Syntax Tree* (AST). Deteksi dilakukan dengan melakukan komparasi terhadap sub-tree yang telah dihasilkan pada proses transformasi kode ke AST. Beberapa kakas bantu juga telah menerapkan teknik ini. Selain itu pendekatan dengan AST juga berhasil mendeteksi kode klon tipe 1,2, dan 3 (Rattan & Kaur, 2016) (Koschke, Falke, & Frenzel, 2006) (Jiang et al., 2007).
- **Teknik Berbasis *Program Dependence Graph* (PDG) :** pada pendekatan ini, kode akan ditransformasikan menjadi PDG. Dengan PDG, kode program direpresentasikan dengan graf. Deteksi dilakukan dengan melakukan komparasi terhadap sub-graf yang telah dihasilkan pada proses transformasi kode. Pendekatan dengan PDG dapat digunakan untuk

menginsvestigasi kode klon tipe 4 atau semantik klon. (Rattan & Kaur, 2016) (Priyambadha & Rochimah, 2014).

- **Teknik Berbasis Metrik :** pada pendekatan ini, kode tidak dibandingkan secara langsung. Untuk melakukan deteksi kode klon, digunakan beberapa matrik dari kode. Hasil dari matrik-matrik tersebut akan dibandingkan untuk menemukan kemungkinan kode klon (Rattan & Kaur, 2016) (Bansal & Tekchandani, 2014).

Pada beberapa penelitian membuat pendekatan deteksi kode klon secara spesifik hanya untuk tipe kode klon tertentu. Seperti penelitian yang dilakukan Priyambadha (Priyambadha & Rochimah, 2014). Mereka membuat pendekatan yang hanya berfokus pada kode klon tipe 4 atau semantik klon (Priyambadha & Rochimah, 2014).

2.4.2. Deteksi SQL Klon

SQL atau *Structured Query Language* merupakan bahasa yang digunakan untuk mengakses basis data. SQL tidak sama dengan bahasa pemrograman yang lain misal C, Java, PHP dan sebagainya. Karena SQL merupakan bahasa deklaratif (Akbarnejad, 2010). SQL memiliki struktur bahasa yang berbeda dengan bahasa pemrograman pada umumnya seperti C, Java, dsb.

Aplikasi berbasis web merupakan aplikasi yang biasanya dibangun dengan beberapa bahasa pemrograman sekaligus. Penelitian deteksi duplikat kode pada bahasa pemrograman yang dipakai pada web, misalnya HTML (Cordy & Dean, 2004), Javascript (Cheung et al., 2016) dan CSS (Mazinanian et al., 2014), pernah dilakukan. Tetapi penelitan yang secara spesifik membahas SQL klon masih sangat sedikit. Padahal SQL sangat erat kaitannya dengan aplikasi web.

Deteksi duplikasi kueri SQL pada web MVC telah muncul pada kerangka kerja Django (Dobarkod, 2017). Deteksi duplikasi SQL tersebut masih sederhana, hanya mengakomodasi duplikat SQL yang memiliki struktur sama persis (*exact clone*). Selain itu, deteksi duplikat kueri ini hanya melakukan komparasi kueri pada setiap HTTP Request, sehingga hanya melakukan deteksi pada kueri-kueri yang dieksekusi pada HTTP Request. Tidak komprehensif pada keseluruhan kode.

Tetapi, deteksi duplikat kueri pada Django tersebut merupakan bukti bahwa duplikasi SQL perlu dilakukan.

2.4.3. Metode Usulan SQL Klon

Walaupun penelitian di area kode klon banyak dan berkembang. Tetapi penelitian pada SQL klon masih sedikit. Penelitian terkait pembahasan kemiripan SQL dapat ditemukan pada penelitian di area rekomedasi kueri (Eirinaki, Abraham, Polyzotis, & Shaikh, 2014) dan serangan injeksi SQL (Das, Sharma, & Bhattacharyya, 2010).

Pada penelitian Das dkk. (Das et al., 2010), mereka mengusulkan pendeteksian serangan SQL injeksi dengan metode inkremental. Das dkk. menggunakan dua algoritma yang mereka sebut sebagai *exact matching* dan *approximate matching* untuk mendeteksi serangan SQL injeksi. Pertama, mereka melakukan *exact matching* antara kueri dengan rule yang ada. Jika tidak terdeteksi pada *exact matching* maka akan dilakukan *approximate matching*.

Pada penelitian ini, tidak menggunkan pendekatan seperti pada Das dkk. tetapi ide penggunaan inkremental matching akan diadaptasi pada penelitian ini untuk mendeteksi tipe-tipe SQL klon. Metode usulan akan dibahas lebih detail pada bab selanjutnya.

2.4.4. *Cosine Similarity*

Penggunaan cosine similarity untuk menghitung kemiripan pada dua fragmen kode pernah dilakukan pada beberapa penelitian (Eirinaki et al., 2014) (Sheneamer & Kalita, 2016). Pada penelitian yang dilakukan oleh Sheneamer (Sheneamer & Kalita, 2016) menunjukkan penggunaan metode cosine similarity efektif untuk mendeteksi kode klon tipe 3.

Cosine similarity merupakan metode untuk menghitung kemiripan antara dokumen/kalimat/string dengan mengubah setiap kata pada suatu string menjadi sebuah vector. Dari vector tersebut akan diketahui kedekatan atau kemiripan dari kedua string tersebut. Berikut rumus cosine similarity (Sheneamer & Kalita, 2016).

$$CosSim(q1, q2) = \frac{\overline{q1} \,.\, \overline{q2}}{|\overline{q1}| x |\overline{q2}|} \quad (2.1)$$

Dari rumus diatas q1 dan q2 adalah sepasang string yang dibandingkan. Kemudian $\overline{q1}$ dan $\overline{q2}$ merupakan bentuk vector dari q1 dan q2.

2.5. Evaluasi Hasil Deteksi Klon

Pada penelitian ini, untuk mengevaluasi hasil deteksi klon yang dideteksi oleh sistem digunakan perhitungan akurasi, *precision*, *recall* dan *f-measure*. Evaluasi akurasi ini digunakan untuk mengetahui ketepatan sistem dalam mendeteksi klon SQL dibandingkan dengan dataset SQL skrip yang telah terlabeli klon SQL.

Dalam penelitian ini, satu dataset merepresentasikan satu project sistem berbasis MVC. Dan evaluasi akurasi dilakukan pada setiap project atau dataset. Berikut ini merupakan cara evaluasi akurasi hasil deteksi klon SQL:

$$\text{Akurasi Deteksi Klon SQL} = \frac{Jumlah\ klon\ SQL\ yang\ dideteksi\ dengan\ benar}{Jumlah\ klon\ SQL\ dataset} \quad (2.2)$$

Dimana:

- **Jumlah klon SQL yang dideteksi dengan benar** : merupakan jumlah klon yang dideteksi oleh sistem yang sesuai dengan dataset.
- **Jumlah klon SQL dataset :** merupakan jumlah semua data klon SQL pada dataset.

Perhitungan *precision*, *recall* dan *f-measure* mengacu pada penelitian Hotta (Hotta, Yang, Higo, & Kusumoto, 2014). P merupakan data referensi klon sedangkan T merepresentasikan sistem.

$$\text{Precision (P, T)} = \frac{|DetectedRefs(P,T)|}{|Cands\ (P,T)|} \quad (2.3)$$

$$\text{Recall (P, T)} = \frac{|DetectedRef(P,T)|}{|Refs\ (P)|} \quad (2.4)$$

Dimana:

- *DetectedRefs(P,T) :* bagian refrenesi dalam P yang terdeteksi benar oleh T.
- *Cands (P,T) :* bagian dari kandidat klon T yang terdeteksi dari P.

- *Ref (P) :* semua referensi pada P, dimana berisfat independen.

Nilai *F-Measure* menunjukkan harmonisasi antara nilai *precision* dan *recall.* Semakin tinggi nilai F-Measure menunjukkan semakin tinggi nilai keseimbangan antara nilai *precision* dan *recall.* Nilai F-Measure dapat dihitung dengan persamaan berikut:

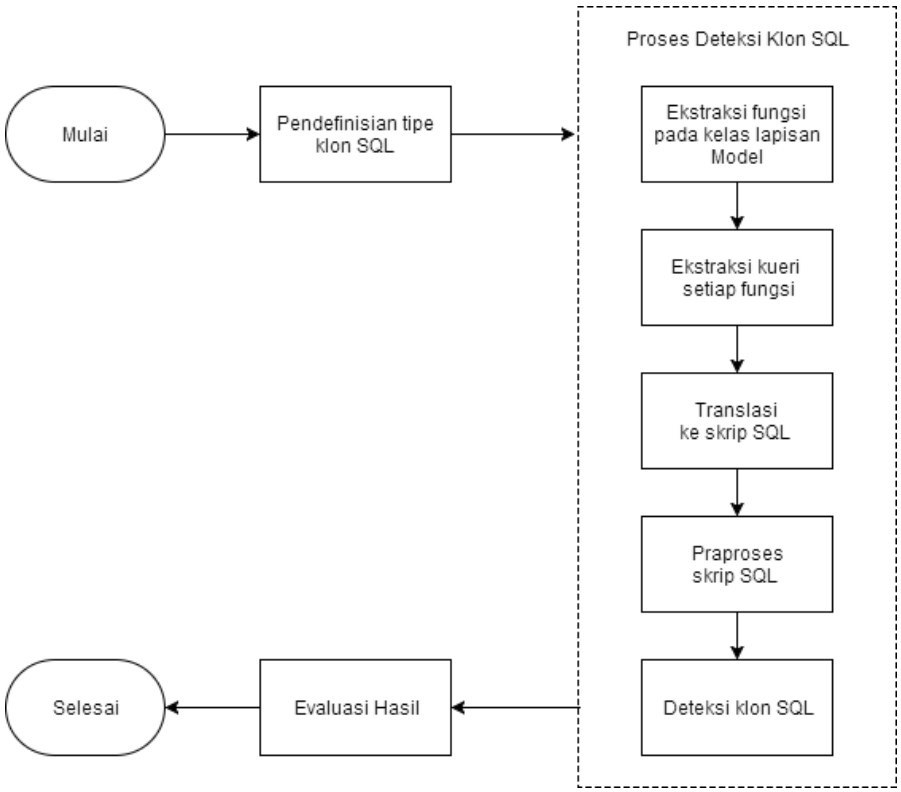
$$\mathrm{F-Measure\ (P,T)} = \frac{2*Precision\ (P,T)*Recall\ (P,T)}{Precision(P,T)+Recall(P,T)} \quad (2.5)$$

BAB 3
METODOLOGI PENELITIAN

Pada Bab ini akan dijelaskan Metodologi Penelitian. Langkah-langkah yang akan dilakukan pada penelitian ini dapat dilihat pada Gambar 3.1.

3.1. Tahapan Penelitian

Berdasarkan rumusan masalah, maka tahapan penelitian ini dibagi menjadi tiga bagian utama yaitu proses pendefinisian tipe klon SQL, proses deteksi klon SQL, dan yang terakhir adalah evaluasi hasil deteksi klon SQL. Sebelum proses deteksi klon SQL hal pertama dilakukan adalah mendefinisikan klon SQL yang mencakup pada tipe-tipe klon pada SQL. Definisi tipe-tipe klon SQL yang dibuat berpengaruh pada pendekatan pendeteksian klon SQL. Proses evaluasi hasil deteksi digunakan untuk memvalidasi kebenaran deteksi pada klon SQL. Tahapan penelitian tersebut ditunjukkan pada Gambar 3.1 berikut.



Gambar 3.1. Tahapan Penelitian

Berdasarkan Gambar 3.1. Proses deteksi klon SQL diawali dengan melakukan ekstraksi fungsi-fungsi pada lapisan model. Kemudian melakukan

ekstraksi kueri pada setiap fungsi. Setelah kueri-kueri dalam bentuk *raw* SQL atau *query builder* berhasil diekstraksi, langkah selanjutnya adalah melakukan translasi ke SQL skrip dan dilakukan praproses pada SQL skrip. Langkah selanjutnya adalah melakukan deteksi klon SQL. Penjelasan lebih lanjut tahapan penelitian ini dijelaskan pada subbab berikut.

3.1.1. Pendefinisian Klon SQL

Untuk proses pendefinisian tipe klon pada skrip SQL ini, diambil dari dua dasar yaitu definisi kode klon dan hasil penelitan di area similaritas skrip SQL. Penelitian di area similaritas skrip SQL mencakup mutasi SQL skrip dan sistem rekomendasi untuk kueri SQL.

Pendefinisian tipe klon berdasarkan tipe kode klon yang ada telah dilakukan oleh Störrle dkk (Storrle, 2013). Mereka mendefinisikan tipe klon pada UML dengan menyesuaikan definisi tipe klon pada area kode. Strolle dkk mendefinisikan empat tipe klon yaitu tipe A,B,C dan D yang merupakan representasi dari kode klon 1,2,3, dan 4.

Proses pendefinisian SQL klon mempertimbangkan dari hasil investigasi dari mutasi skrip SQL. Mutasi SQL dipilih karena dengan proses mutasi SQL skrip akan diketahui tingkat kemiripan dari SQL skrip. Cukup banyak penelitian dan tools yang membahas mutasi pada skrip SQL (Tuya, Suárez-cabal, & Riva, 2006). Selain mutasi skrip SQL, penelitian yang membahas similaritas dari SQL skrip adalah penelitian di area sistem rekomendasi untuk kueri SQL (Eirinaki et al., 2014).

Dasar penggunaan mutasi SQL ini adalah proses mutasi dapat digunakan untuk menghasilkan data untuk melakukan testing pada algoritma deteksi kode klon. Mutasi kode dilakukan karena data set yang tersedia untuk penelitian kode klon masih sangat sedikit (Stephan, 2014). Kode program akan dimutasi menjadi kode kode yang mengakomodasi setiap tipe dari kode klon (Stephan, 2014). Pada penelitian ini mencoba untuk membalik pendekatan tersebut dengan menggunakan hasil mutasi untuk mendefinisikan tipe klon pada SQL skrip.

Proses percobaan mutasi SQL menggunakan kakas bantu dari penelitian Tuya (Tuya et al., 2006). Masukan dari kakas bantu tersebut adalah sebuah kueri dan keluaran dari kakas bantu adalah hasil mutasi kueri dari kueri awal. Proses percobaan mutasi kueri ini hanya dilakukan untuk *statement* SELECT. Klausa yang digunakan pada percobaan ini meliputi klausa SELECT, FROM, WHERE, GROUP BY, HAVING, GROUP BY, dan fungsi agregat. Hasil dari percobaan mutasi SQL dapat disimpulkan sebagai berikut:

- Penggunaan distict pada klausa select dideteksi sebagai query yang ekuivalen jika klausa yang lain sama persis.
- Pada proses mutasi penggunaan prefiks nama tabel dan kolom pada select juga digunakan.
- Mengganti value dari jenis klausa filter. Missal: **`where nrp='51152983'`** menjadi **`where nrp=90`**
- Jumlah kolom hasil mutasi SQL selalu sama dengan jumlah kolom pada skrip SQL asal.

Hasil percobaan mutasi SQL tersebut dapat dijadikan acuan dalam proses pendefinisian SQL klon atau pada proses praproses skrip SQL.

Sebelum mengidentifikasi definsi tipe-tipe klon pada SQL. Terlebih dahulu didefinisikan ukuran kemiripan dari SQL kueri. Ukuran kemiripan SQL yang digunakan sebagai dasar pada penelitian ini mengacu pada penelitian Eirinaki (Eirinaki et al., 2014). Dua skrip SQL dianggap mirip jika dua skrip SQL melakukan kueri pada tabel dan atribut yang sama dengan menggunakan klausa kondisi yang berbeda. Definisi kemiripan tersebut sebagai dasar pendefinsian klon SQL.

2.5.1.1. Definisi 1: SQL Klon Tipe A

Sepasang SQL kueri dinyatakan masuk pada tipe A jika sepasang query tersebut memiliki tulisan yang sama persis. Kriteria sama persis adalah sepasang query memiliki klausa SELECT, klausa FROM, dan klausa kondisi yang identik. Selain itu memiliki nama dan urutan yang sama persis. Kecuali pada berbedaan spasi. Definisi diatas mengacu pada definisi kode klon tipe 1 atau eksak klon.

Contoh 1:

`select nrp,nama` `from mahasiswa`	`select nrp,nama` `from mahasiswa`

Contoh 2:

`select nrp,nama`
`from mahasiswa`
`where nrp = '5115201006'`
	`select nrp,nama`
	`from mahasiswa`
	`where nrp = '5115201007'`

Semua bentuk input kondisi akan diabaikan. Seperti **`nrp = '5115201006'`** dan **`nrp = '5115201007'`** akan diabaikan menjadi nrp = 'X';

Dari Contoh 1 dan 2 memang akan menampilkan hasil data yang berbeda. Pada Contoh 1 data yang dihasilkan akan sama. Sedangkan pada Contoh 2, data yang ditampilkan berbeda. Tetapi pada Contoh 2 memiliki fungsi yang sama yaitu menampilkan data spesifik yang berbeda hanya pada nilai variabel argumennya.

2.5.1.2. Definisi 2: SQL Klon Tipe B

Sepasang SQL kueri dinyatakan masuk pada tipe B jika sepasang kueri memiliki klausa SELECT, klausa FROM, dan klausa kondisi yang sama. Tetapi memiliki urutan dan penamaan (alias) yang berbeda di kolom dan tabel pada klausa SELECT, klausa FROM, dan klausa kondisi.

Definisi diatas mengacu pada definisi kode klon tipe 2 yaitu klon struktur, memiliki struktur yang sama, perubahan ada pada penamaan (identifier), literal, tipe data, layout dan komentar.

SQL memungkinkan adanya perubahan penamaan field atau tabel karena SQL mengenal penggunaan alias. Penulisan alias pada SQL ada dua acara yaitu menggunakan kata kunci AS atau langsung menuliskan nama alias setelah nama field atau tabel.

```
select nrp as nomor_induk, nama nama_mhs from mahasiswa mhs
```

Tabel 3.1. berikut contoh dari klon SQL tipe B :

Tabel 3.1. Contoh klon SQL tipe B

No	Q1	Q2
1	Select nama, nrp from mahasiwa	Select nrp, nama from mahasiswa
2	Select nama, nrp, id_fakultas from mahasiswa, fakultas where mahasiswa.id_fakultas= fakultas.id_fakultas	Select nrp, nama as nama_mhs, id_fakultas as kode_fakultas from fakultas, mahasiswa where fakultas.id_fakultas= mahasiswa.id_fakultas

Skrip SQL pada Q1 dan Q2 merupakan contoh dari klon tipe B. Pada pasangan SQL nomor 1, menunjukkan perbedaan letak urutan dari kolom pada klausa select. Sedangkan pada pasangan SQL nomor 2, menunjukkan perbedaan letak urutan pada klausa SELECT, FROM, dan WHERE. Selain itu, terdapat penggunaan nama alias pada klausa SELECT.

2.5.1.3. Definsi 3: SQL Klon Tipe C

Sepasang SQL kueri dinyatakan masuk pada tipe C jika sepasang kueri memiliki syarat sebagai berikut:

- Memiliki klausa SELECT yang sama (tanpa memperhatikan perbedaan nama dan urutan)
- Pada klausa FROM memiliki tabel yang sama tetapi menggunakan cara join yang berbeda, atau minimal memiliki satu tabel yang sama.
- Atau memiliki klausa kondisi yang berbeda.

Pada pendefinisian ini, definisi dari kode klon tipe 3 perlu disesuaikan. Karena struktur kode pemrograman (C, Java, PHP, dan sebagainya) dengan SQL berbeda. Kode klon tipe 3 mendefinisikan bahwa ada penambahan dan penghapusan fragmen pada kode. Tetapi penghapusan dan penambahan fragmen pada SQL dapat menghasilkan data yang berbeda. Contoh:

```
select nrp, nama from mahasiswa
```

Dengan

```
select nrp, nama, id_fakultas, alamat from mahasiswa
```

Kedua kueri tersebut dianggap berbeda karena akan menampilkan kolom data yang pasti berbeda, pada kueri kedua menampilkan data id_fakultas dan alamat sedangkan pada kueri pertama tidak ada informasi tersebut.

Menurut Eirinaki (Eirinaki, Abraham, Polyzotis, & Shaikh, 2014) jika dua SQL skrip melakukan kueri pada tabel dan atribut yang sama dengan menggunakan klausa kondisi yang berbeda, hal tersebut masih dapat dianggap mirip.

Jadi berdasarkan dari definsi kode klon tipe 3 dan Enaki dkk. penambahan atau penghapusan fragmen hanya bisa dilakukan pada klausa kondisi. Perbedaan dengan kode klon, penambahan dan penghapusan fragmen bisa dilakukan dimana saja pada kode fragmen, tetapi pada SQL penghapusan dan penambahan fragmen hanya bisa dilakukan pada klausa kondisi agar SQL tersebut masih dapat dikatakan mirip.

Contoh:

select nrp,nama **from mahasiswa** **where nrp = '5115201006'**	**select nrp,nama** **from mahasiswa** **where nrp like '5115%' and** **id fakultas='08'**

Pada contoh di atas menunjukkan bahwa kedua skrip SQL manampilkan kolom yang sama tetapi menghasilkan data yang berbeda karena adanya perbedaan pada klausa kondisi. Hal tersebut dapat dikategorikan klon tipe C.

Syarat minimal sama satu tabel pada klausa FROM berasal dari adanya kemungkinan tabel yang digunakan untuk tujuan kondisi. Seperti ditunjukkan pada contoh dibawah ini.

Contoh:

- ```
  Select nrp, nama
  From mahasiswa
  ```

- ```
  Select nrp, nama
  From mahasiswa, fakultas
  Where mahasiswa.id_fakultas = fakultas.id_fakultas
  And fakultas.nama_fakultas = 'FTIF'
  ```

Jika syarat kondisi FROM harus dari tabel yang sama, maka contoh diatas akan tidak dianggap sebagai klon. Pada contoh diatas, penambahan tabel dalam

klausa form tidak mempengaruhi kolom yang ditampilkan. Sehingga syarat minimal dari satu tabel yang sama diharapkan tetap dapat mendeteksi contoh kasus diatas sebagai klon tipe C.

- **Perbandingan Cross Join dan JOIN**

Pada klausa FROM memungkinkan adanya penggabungan (join) antar tabel untuk mendapatkan data yang relevan. Cara penggabungan tabel tersebut dapat dilakukan dengan beberapa cara, yaitu cross join dan menggunakan kata kunci yang disediakan oleh SQL, yang meliputi JOIN, LEFT JOIN, RIGHT JOIN.

Terkait dengan adanya penggabungan tabel dan syarat SQL klon tipe C. Maka yang diidentifikasi adalah nama tabel-tabel yang ada pada klausa FROM tanpa memperhitungkan perebedaan cara penggabungan tabel (join). Hal ini dilakukan untuk mengetahui terpenuhinya syarat SQL klon tipe C. Yaitu pada klausa FROM minimal memiliki satu tabel yang sama.

Jika dua SQL berasal dari tabel yang sama, maka selanjutnya dilakukan perbandingan klausa FROM secara keseluruhan untuk mengetahui kesamaan kalusa FROM. Jika memiliki bentuk klausa FROM yang berbeda maka dapat dimasukkan kedalam klon SQL tipe C. Seperti contoh pada Tabel 3.2. berikut:

Tabel 3.2. Contoh klon SQL tipe C dengan perbedaan tipe join

No	Q1	Q2

1	`select nrp, nama_fakultas`
	`from mahasiswa, fakultas`
	`where mahasiswa.id_fakultas = fakultas.id_fakultas`
		`select nrp, nama_fakultas`
		`from mahasiswa JOIN ON fakultas`
		`ON mahasiswa.id_fakultas = fakultas.id_fakultas`

2	`select nrp, nama_fakultas`
	`from mahasiswa LEFT JOIN ON fakultas`
	`ON mahasiswa.id_fakultas = fakultas.id_fakultas`
		`select nrp, nama_fakultas`
		`from mahasiswa RIGHT JOIN ON fakultas`
		`ON mahasiswa.id_fakultas = fakultas.id_fakultas`

Pada Tabel 3.2., pasangan SQL (Q1,Q2) pada nomor 1 memiliki struktur klausa FROM yang berbeda walaupun memiliki tabel yang sama yaitu tabel **`mahasiswa`** dan **`fakultas`**. Q1 menggunakan *cross join* sedangkan Q2 menggunakan kata kunci JOIN. Walaupun Q1 dan Q2 menghasilkan data yang

sama, Q1 dan Q2 akan dikategorikan klon SQL tipe C karena memiliki struktur yang berbeda.

Pasangan SQL (Q1,Q2) nomor 2 memiliki perbedaan kalusa FROM pada kata kunci LEFT JOIN dan RIGHT JOIN. Perbedaan struktur klausa FORM tersebut dikategorikan klon SQL tipe C.

2.5.1.4. Definisi 4: SQL Klon Tipe D (Klon Parsial)

Sepasang SQL kueri dinyatakan masuk pada tipe D jika sepasang kueri termasuk klon tipe A, B, atau C. Tetapi salah satu atau kedua kueri tersebut merupakan bagian dari sebuah kueri lain. Hal ini dapat ditemukan pada contoh kasus sub-kueri atau penggabungan dua kueri (UNION) pada Tabel 3.3. berikut ini.

Tabel 3.3. Contoh klon SQL tipe D

No	Q1	Q2
1	select id_fakultas from mahasiswa where nrp = '094801'	select nrp,nama from mahasiswa where id_fakultas = (select id_fakultas from mahasiswa where nrp = '9205')
2	select nrp, nama from mahasiswa	select nrp, nama from mahasiswa **UNION** select nrp, nama from alumni

Pada contoh nomor 1 di Tabel 3.3. menunjukkan bahwa sebagian skrip SQL (sub-kueri) pada Q2 merupakan klon dari Q1. Sedangkan pada contoh nomor 2, Q1 dan Q2 merupakan klon tipe D karena bagian skrip SQL Q2 (bagian dari klausa UNION) adalah klon dari Q1.

3.1.2. Proses Deteksi Klon SQL

Sebelum membahas tentang motode deteksi klon SQL. Terlebih dahulu akan dibahas hasil percobaan kecil dari deteksi duplikat SQL dengan menggunakan kakas bantu deteksi kode klon. Perkakas bantu yang ada saat ini, masih belum mengakomodasi deteksi klon pada SQL skrip. Pada penelitian ini telah mencoba mendeteksi SQL skrip pada perkakas CCFinderX (Kamiya et al., 2002). Perkakas CCFinderX adalah perkakas deteksi klon berbasis token. Perkakas CCFinderX

tidak menyediakan deteksi khusus untuk SQL. Uji coba yang dilakukan menggunakan skrip SQL sebagai inputan. Skrip SQL tersebut terlebih dahulu dimutasi menjadi beberapa skrip SQL yang mirip. Misalnya, mengubah urutan kolom dan tabel, menambahkan alias, menambah klausa pada filter. Berikut ini kesimpulan dari percobaan yang telah dilakukan:

- Kode fragmen SQL yang terlalu pendek seperti `select * from nama_tabel` tidak dapat terdeteksi. Karena pada kakas bantu ada batas minimal dari jumlah token yang digunakan untuk deteksi.
- Jika urutan field pada klausa select, from, dan where berbeda walaupun sebenarnya memiliki fungsionalitas yang sama, hal tersebut gagal dideteksi.
- Gagal deteksi terjadi jika ada penggunaan nama alias pada table dan field pada kueri yang identik.
- Tidak mengenali keyword-keyword khusus pada SQL skrip. Misalkan SELECT, FROM, WHERE, dan sebagainya.
- Deteksi bisa berjalan jika dua query memiliki sequence kata yang benar-benar sama (urutan kolom, tabel, dan klausa where) dengan melakukan highlight pada sequence yang sama.

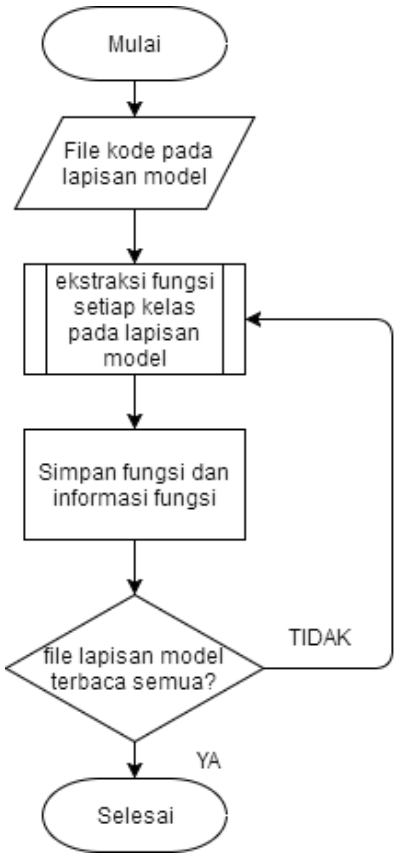
Dari hasil percobaan tersebut, maka perlu didefinisikan metode atau langkah-langkah yang dapat digunakan untuk mendeteksi klon pada SQL. Dan sekaligus dapat mendeteksi tipe-tipe klon yang telah didefinisikan sebelumnya.

Proses deteksi klon SQL pada penelitian ini terdiri dari beberapa langkah yaitu (1) Ekstraksi Fungsi Kelas pada Lapisan Model, (2) Ekstraksi Kueri setiap Fungsi, (3) Translasi SQL Skrip, (4) Praproses SQL Skrip, dan (5) Deteksi Klon SQL Skrip. Detail dari langkah-langkah tersebut dapat dijabarkan sebagai berikut.

3.1.2.1. Ekstraksi Fungsi Kelas pada Lapisan Model

Pada proses yang pertama, semua kode pada kerangka kerja akan MVC akan diabaikan kecuali kode-kode pada lapisan model. Semua fungsi pada kelas di lapisan model diekstraksi. Proses ekstraksi ini dilakukan dengan memparsing kode pada kelas-kelas lapisan model. Informasi yang disimpan pada ekstraksi fungsi

adalah kode fungsi, nama fungsi, dan kelas fungsi. Proses ekstraksi dilakukan sampai semua file dalam lapisan model terbaca. Langkah ekstraksi fungsi lapisan model ditunjukkan pada Gambar 3.2. berikut.



Gambar 3.2. Proses Ekstraksi Fungsi pada Lapisan Model

3.1.2.2. Ekstrakasi Kueri Setiap Fungsi

Langkah selanjutnya adalah melakukan ekstraksi kueri pada setiap fungsi. Ekstraksi kueri ini meliputi kueri yang dibangun dengan *raw* SQL dan *query builder*. Proses ekstraksi kueri ini dilakukan dengan pendekatan statis. Pendekatan statis adalah pendekatan tanpa menjalankan kode program. Proses ekstraksi kueri mengacu dari pola-pola kueri dari hasil penelitian Anderson (Anderson & Hills, 2017.). Ekstraksi kueri pada peneltian ini mengabaikan alur kontrol.

Cara mendeteksi kueri pada setiap fungsi pada lapisan model adalah mendeteksi kata kunci kueri. Karena obyek penelitian ini menggunakan kerangka kerja CodeIgniter maka kata kunci untuk kueri adalah **`$this->db->[get,select,where,..]`** dan untuk kueri raw SQL pada CodeIgniter adalah

`$this->db->query(…).` Karena adanya dua bentuk penulisan kueri yaitu raw SQL dan kueri builder, maka dalam penelitian ini dibedakan cara ekstraksi kueri dari dua bentuk kueri tersebut.

- **Ekstraksi kueri *raw* SQL**

Raw SQL pada CodeIgniter diidentifikasi dengan pemanggilan fungsi **`$this->db->query(…).`** Pemanggilan fungsi tersebut dapat diikuti dengan argumen berupa skrip SQL atau dengan variabel yang berisi skrip SQL.

Tabel 3.4. Contoh ekstraksi kueri *raw* SQL

No	Fungsi pada lapisan model	Kueri teridentifikasi

A	function get_bank(){
	$bank =
	$this->db->query("select * from bank");
	return $bank->result();
	}
		$this->db->query("select * from bank");

B	function get_account(){
	$sql = "select id_bank, no_rekening, nama, saldo from bank";
	$query = $this->db->query($sql);
	return $query->result();
	}
		$sql = "select id_bank, no_rekening, nama, saldo from bank";
		$query = $this->db->query($sql);

Pada Tabel 3.4. nomor A, skrip SQL langsung dijadikan argumen pada pemanggilan fungsi raw SQL **`$this->db->query(…).`** Pada contoh B, skrip SQL disimpan dalam sebuah variabel kemudian variabel tersebut dijadikan argumen pada pemanggilan fungsi raw SQL. Untuk kasus pada contoh B, akan diidentifikasi apakah argumen SQL berbentuk skrip SQL atau variabel. Jika berbentuk variabel maka akan diidentifikasi variabel argument tersebut kemudian digabungkan dengan *statement* pemanggil raw SQL. Ekstraksi kueri dengan penggunaan variabel ditunjukkan pada hasil kueri teridentifikasi pada contoh B.

- **Ekstraksi kueri *query builder***

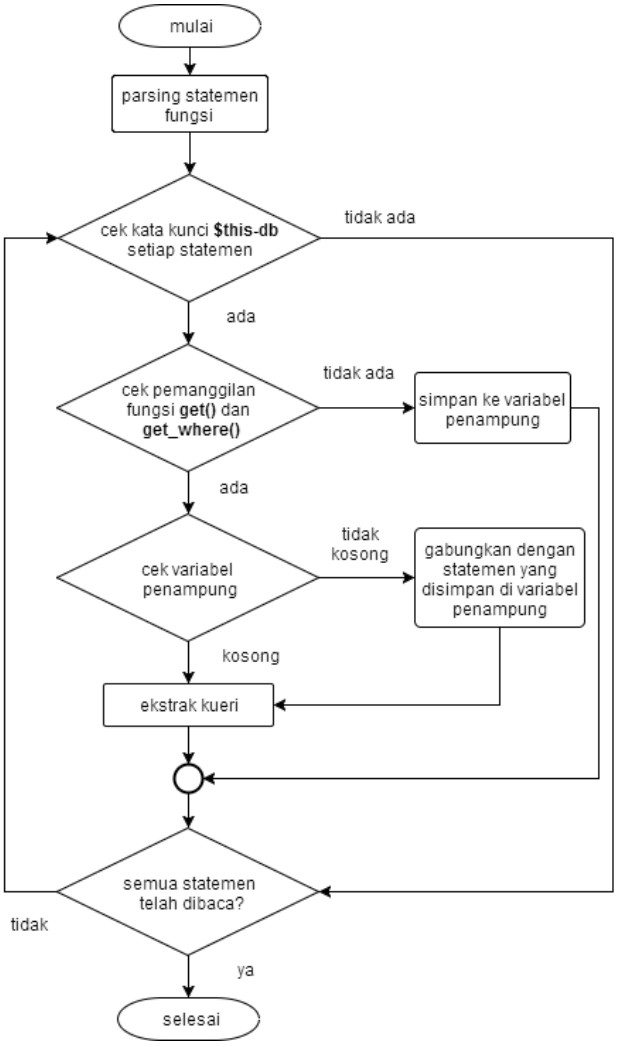
Pembentukan kueri query buider lebih dinamis dibandingkan dengan raw SQL. pemanggilan fungsi pada query builder lebih banyak dibandingkan dengan raw SQL seperti **`$this->db->[get,get_where,select,from,where,..].`** Contoh bentuk kueri dengan *query builder* ditunjukkan pada Tabel 3.5. berikut ini:

Tabel 3.5. Contoh bentuk *query builder*

No	*Query Builder*
A	`$query = $this->db->select('*')->from('admin')->where('id','84923642')->get();`
B	`$this->db->select('*')->from('admin')->where('id','84923642');` `$query = $this->db->get();`
C	`$query = $this->db->get_where('admin',array('id'=>'84923642'));`

D	`$this->db->select('*');`
	`$this->db->from('admin');`
	`$this->db->where('id','84923642');`
	`$this->db->get();`

Bentuk *statement* SELECT pada query builder selalu diakhiri dengan pemanggilan fungsi $**this->db->get()** atau **$this->db->get_where().** Sehingga, dua pemanggilan fungsi tersebut digunakan untuk mengidentifikasi kueri yang dibentuk dengan *query builder*. Langkah-langkah yang dilakukan untuk mengekstraksi query builder pada penelitian ini ditunjukkan pada Gambar 3.3 berikut:



Gambar 3.3. Langkah ekstraksi *query builder*

Langkah-langkah ekstraksi kueri builder pada Gambar 3.3 dijelaskan secara sederhana sebagai berikut:

1. Mengekstraksi setiap *statement* dengan menggunakan parser.
2. Setiap *statement* dalam fungsi dilakukan identifikasi pemanggilan query builder dengan mengidentifikasi kata kunci **`$this->db.`** Jika pada suatu *statement* yang terdapat kata kunci **`$this->db`** tidak ditemukan pemanggilan fungsi get() atau get_where(), maka *statement* tersebut akan disimpan dalam suatu array.
3. Jika ditemukan pemanggilan fungsi get() atau get_where(), selanjutnya akan dilakukan pengecekan apakah array penampung *statement* **`$this->db`** kosong atau tidak. Jika isi array kosong, maka satu *statement* yang ada pemanggilan fungsi get() atau get_where() akan diekstraksi sebagai satu kueri. Contoh:

```
$query = $this->db->select('*')->from('admin')
         ->where('id','84923642')->get();
```

   jika isi array tidak kosong, maka *statement* yang disimpan dalam array sebelumnya akan digabungkan dengan *statement* yang ada pemanggilan fungsi get() atau get_where(), seperti contoh pada Tabel 3.6:

Tabel 3.6. Contoh ekstraksi kueri *query builder*

No *statement*	***Statement***
a	`$this->db->select('*');`
b	`$this->db->from('admin');`
c	`$this->db->where('id','84923642');`
d	`$this->db->get();`

   Tabel 3.6. menunjukkan hasil parsing setiap *statement* pada *query builder*. Pada saat ekstraksi, *statement* a,b,c akan disimpan dalam sebuah variabel penampung, dan ketika ditemukan *statement* d yang terdapat pemanggilan fungsi get(), maka akan dilakukan penggabungan *statement* yang disimpan pada variabel penampung (a,b,c) dengan *statement* d. sehingga hasil akhir ekstraksi kueri adalah *statement* a,b,c,d.

Setelah kueri teridentifikasi, langkah terakhir pada proses ekstraksi kueri adalah mengganti variabel atau pemanggilan fungsi lain yang berhubungan dengan kueri menjadi literal “1”. Contoh pada hasil kueri yang teridentifikasi sebelumnya.

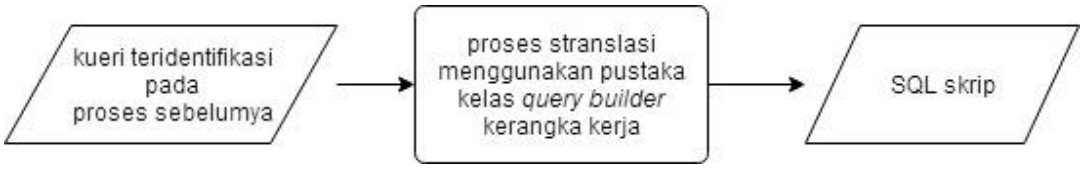
```
  $query = "select rekening, atas_nama, saldo from account where
rekening = ".$param['rek'];
  $this->db->query($query);
```

Menjadi

```
  $query = "select rekening, atas_nama, saldo from account where
rekening = "."1";
  $this->db->query($query);
```

3.1.2.3. Translasi Skrip SQL

Langkah selanjutnya adalah proses translasi skrip SQL. Pada proses ini kueri yang berhasil diidentifikasi pada proses sebelumnya akan ditranslasi ke skrip SQL. Proses translasi ini memanfaatkan kelas *query builder* pada kerangka kerja web MVC yang dipilih. Gambaran umum dari proses trasnlasi skrip SQL ditunjukkan pada Gambar 3.4. berikut.



Gambar 3.4. Proses translasi skrip SQL

Pemanfaatan kelas-kelas *query builder* akan sangat membantu pada proses transalasi selain itu hasil dari translasi kueri ke SQL skrip lebih akurat. Pada Tabel 3.7. berikut menunjukkan contoh hasil translasi dari kueri ke skrip SQL.

Tabel 3.7. Contoh translasi skrip SQL

Kueri Teridentifikasi	Hasil skrip SQL
`$this->db->select ('id_bank','no_rekening','nama'); $this->db->get('bank');`	**`select id_bank, no_rekening, nama from bank`**

3.1.2.4. Praproses SQL Skrip

Praproses SQL skrip dilakukan untuk memudahkan proses deteksi dengan menyamakan beberapa bentuk SQL. Praproses ini mengakomodasi tipe-tipe klon SQL yang telah didefinsikan sebelumnya. tujuan dari praproses ini adalah untuk memudahkan pada proses deteksi klon SQL dan menambah akurasi dari deteksi. Proses praproses SQL skrip sebagai berikut:

- Mengganti semua value atau literal dengan karakter khusus.

  Praproses ini digunakan untuk menyamakan semua bentuk value atau literal pada skrip SQL. Dengan penggantian value ini, semua value atau literal akan diintrepretasi menjadi bentuk yang sama.

```
select rekening, atas_nama, saldo
from account
where id_transaction = 5
```

  Menjadi

```
select rekening, atas_nama, saldo
from account
where id_transaction = 1
```

- Menghilangkan semua nama alias pada field dan tabel.

  Praproses ini digunakan untuk menghilangkan nama alias pada kolom (*field*) dan tabel. Dengan penghilangan nama alias ini akan memudahkan pada saat proses komparasi kolom skrip SQL. Praproses ini mengakomodasi klon SQL tipe B.

```
select rekening nomor, atas_nama as pemilik, saldo
from account as tabungan
```

  menjadi

```
select rekening, atas_nama, saldo
from account
```

- Menghilangkan semua prefiks

  Praproses ini digunakan untuk menghilangkan prefiks kolom disemua klausa pada *statement* SELECT. Penghilangan prefiks ini akan membantu saat komparasi nama kolom pada proses deteksi klon SQL.

```
select ac.rekening nomor, ac.atas_nama as pemilik, ac.saldo
from account ac, nasabah ns
where ac.rekening = ns.rekening and ns.id_kota = 32;
```

menjadi

```
select rekening, atas_nama, saldo
from account, nasabah
where rekening = rekening and id_kota = 1;
```

- Memisahkan kueri yang menjadi bagian dari subkueri.

  Praproses dilakukan untuk mengakomodasi klon SQL tipe D atau parsial klon. Subkueri pada SQL akan dipisahkan menjadi kueri tersendiri. Hal ini dilakukan agar subkueri dapat dibandingkan dengan skrip SQL yang lain.

```
select nrp,nama
from mahasiswa
where id_fakultas = (
   select id_fakultas
   from mahasiswa
   where nrp = '094900'
)
```

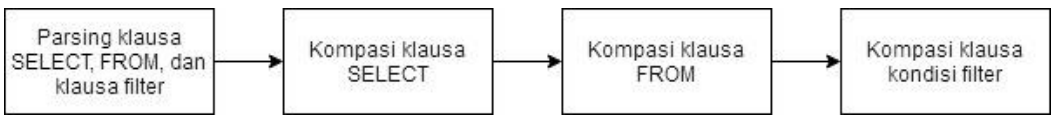
menjadi

```
select id_fakultas
from mahasiswa
where nrp = '094801'
```

3.1.2.5. Deteksi Klon SQL Skrip

Metode usulan untuk pendeteksian klon SQL terinspirasi dari penelitian Das dkk. (Das et al., 2010) dalam pendeteksian serangan injeksi SQL. Pada penelitian ini, tidak secara langsung mengadaptasi penelitian Das dkk. tetapi hanya mengambil ide dari penggunaan komparasi berjenjang (*incremental matching*). Ide komparasi berjenjang tersebut akan diadaptasi pada penelitian ini untuk mendeteksi tipe-tipe SQL klon. Pada proses deteksi, SQL skrip yang berasal dari fungsi yang sama tidak dilakukan deteksi karena kueri-kueri dalam satu fungsi biasanya merupakan variasi dari kueri yang lain.

Berikut ini dijelaskan terkait dengan metode usulan untuk mendeteksi klon pada skrip SQL. Langkah-langkah metode usulan dapat digambarkan pada Gambar 3.5 berikut ini:

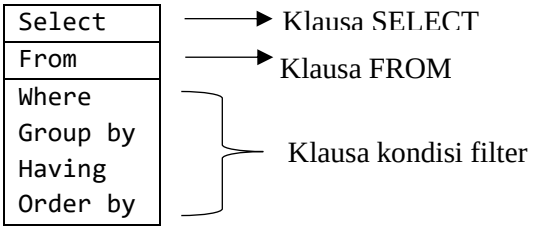


Gambar 3.5. Langkah deteksi klon SQL

Metode usulan untuk mendeteksi klon SQL dibagi menjadi tiga bagian yaitu komparasi klausa SELECT, komparasi klausa FROM, dan komparasi klausa kondisi filter. Pada penelitian ini hanya berfokus pada *statement* SELECT. Berikut detail dari langkah deteksi klon SQL.

A. Parsing Skrip SQL

Pada proses ini, skrip SQL akan diparsing menjadi tiga bagian utama yaitu berdasarkan kata kunci klausa SELECT, FROM, dan klausa kondisi filter (WHERE, GROUP BY, HAVING, ORDER BY). Pembagian SQL skrip menjadi tiga bagian ini terkait dengan definisi SQL klon dan tipe-tipe SQL klon yang telah didefinisikan sebelumnya. Gambar 3.6 berikut adalah gambaran umum bagian parsing SQL skrip:



Gambar 3.6. Parsing skrip SQL

Contoh dari parsing skrip SQL ditunjukkan pada Gambar 3.7. sebagai berikut:

`select rekening, atas_nama, saldo`
`from account`
`where id_transaction = X`
	SELECT	`select rekening, atas_nama, saldo`

	FROM	`from account`
	Klausa filter	`where id_transaction = X`

Gambar 3.7. Contoh parsing skrip SQL

B. Komparasi Klausa SELECT

Proses komparasi klausa SELECT dilakukan dengan dua tahap, yang pertama adalah melakukan komparasi jumlah kolom dan yang kedua adalah komparasi nama kolom. Kedua tahap tersebut dapat dijelaskan sebagai berikut:

1. Komparasi Jumlah Kolom pada SELECT

Pada tahap pertama, dilakukan komparasi jumlah kolom pada klausa SELECT. Jika kueri *q1* memiliki jumlah kolom sama dengan kueri *q2*.

Maka akan dilakukan proses selanjutnya, jika tidak, maka kueri *q1* dan *q2* dianggap bukan klon sesuai dengan definisi yang telah dibuat.

$$Jumlah\ kolom\ q1 = Jumlah\ kolom\ q2$$

Komparasi jumlah kolom pada klausa SELECT dapat dicontohkan pada Tabel 3.8. sebagai berikut:

Tabel 3.8. Komparasi jumlah kolom pada klausa SELECT

No	Kueri	Jml kolom
q1	**Select nama, nrp, id_fakultas** **From mahasiswa**	**3**
q2	**Select nrp, nama, id_fakultas** **From mahasiswa**	**3**
q3	Select nama, nrp From mahasiswa	**2**
q4	**Select nrp, nama, id_fakultas** **From mahasiswa** **Where id_fakultas = '51'**	**3**
q5	Select nrp, nama, alamat, tempat_lahir, tgl_lahir From mahasiswa Where id_fakultas = '51'	**5**

Maka yang dari lima kueri diatas, yang memenuhi kriteria adalah kueri nomor *q1*, *q2*, dan *q4*. Kueri yang memenuhi kriteria tersebut akan melalui proses selanjutnya yaitu komparasi kolom.

2. Komparasi Nama Kolom pada SELECT

Setelah mendapatkan kueri yang memiliki jumlah kolom yang sama, maka akan dilakukan selanjutnya yaitu komparasi kesesuaian nama kolom yang diambil setiap kueri yang memiliki jumlah kolom yang sama. Dari hasil pada proses sebelumnya, maka yang *q1*, *q2*, dan *q4* terpilih karena memenuhi syarat. Komparasi nama kolom pada SELECT ditunjukkan pada Tabel 3.9. berikut:

Tabel 3.9. Komparasi nama kolom pada SELECT

No	Kueri	Kolom
q1	Select nama, nrp, id_fakultas From mahasiswa	**nrp, nama, id_fakultas**
q2	Select nrp, nama, id_fakultas From mahasiswa	**nrp, nama, id_fakultas**
q4	Select nrp, nama, id_fakultas From mahasiswa Where id_fakultas = '51'	**nrp, nama, id_fakultas**

C. Komparasi Klausa FROM

Pada langkah ini, dilakukan investigasi apakah kolom yang diambil merupakan dari tabel yang sama. Disini akan didefinisikan minimal jumlah tabel yang sama adalah satu tabel. Hal ini dilakukan untuk tetap mendeteksi klon tipe C.

Contoh:

- ```
  Select nrp, nama
  From mahasiswa
  ```

- ```
  Select nrp, nama
  From mahasiswa, fakultas
  Where mahasiswa.id_fakultas = fakultas.id_fakultas
  And fakultas.nama_fakultas = 'FTIF'
  ```

Pada contoh diatas, merupakan contoh dari sql klon tipe C. dimana penambahan tabel dalam klausa form tidak mempengaruhi kolom yang ditampilkan.

Jika tabel sama persis maka akan dilanjutkan pada langkah selanjutnya. Jika tidak sama persis tetapi tetap memenuhi syarat jumlah tabel yang sama maka dapat dikategorikan dalam kandidat klon tipe C.

Jika dua SQL dideteksi berasal dari tabel yang sama, maka perlu dilakukan perbandingan klausa FROM secara keseluruhan untuk mengetahui kesamaan bentuk kalusa FROM seperti penggunaan tipe join (cross join, join on). Perbandingan klausa FROM dilakukan dengan menghitung nilai *cosine similarity* antara dua klausa FROM. Jika memiliki bentuk klausa FROM yang berbeda maka dapat dimasukkan kedalam klon SQL tipe C.

Permasalahan khusus:

Permasalahan diatas tidak diterapkan pada klausa select * from nama_tabel. Karena hal tersebut akan membuat deteksi tidak tepat. Contoh:

- Select * from a
- Select * from a,b

Kedua kueri tersebut akan menghasilkan kolom yang berbeda, karena bersumber pada tabel yang berbeda. Misalnya, pada tabel *a* ada 10 klom sedangakan pada tabel *b* ada 5 kolom maka jumlah kolom pada kueri 1 adalah 10 sedangkan pada kueri kedua adalah 15.

D. Klausa Kondisi Filter

Klausa WHERE, GROUP BY, HAVING, ORDER BY didefinisikan sebagai klausa filter. Untuk menangani keberagaman pada klausa kondisi filter, maka akan digunakan fungsi similaritas. Contoh metode similaritas adalah cosine similarity atau edit distance. Dalam kasus ini, lebih dipilih pendekatan dengan menggunakan cosine similarity. Karena cosine lebih berfokus pada kumpulan kata, bukan pada urutan kata atau karakter (Moreau, Yvon, & Cappé, 2008). Nilai cosine memiliki rentang antara 1 sampai 0. Nilai 1 mengindikasikan bahwa kedua klausa sama persis sedangkan 0 mengindikasikan bahwa kedua klausa tidak memiliki kemiripan sama sekali.

Metode *cosine similarity* merupakan metode yang menghitung kemiripan antara dua string atau dokumen dengan mengukur jarak derajat sudut dari vektor string atau dokumen. Sebelum dilakukan komparasi, string atau dokumen ditransformasikan ke dalam bentuk vektor. Terminologi pada *cosine similarity* perlu disesuaikan untuk skrip SQL.

- **Dokumen :** pada deteksi klon SQL, yang dimaksud dokumen adalah skrip SQL.
- **Term :** term adalah setiap keyword Bahasa SQL, nama kolom, nama tabel, variabel literal, dan operator.
- **TF (*Term Frequency*) :** jumlah frekuensi kemunculan term (keyword Bahasa SQL, nama kolom, nama tabel, variabel literal, dan operator) pada dokumen (skrip SQL)

Berdasarkan contoh pada langkah sebelumnya, maka akan dihasilkan nilai *cosine similarity* seperti pada Tabel 3.10. berikut:

Tabel 3.10. Komparasi klausa filter dengan *cosine similarity*

kueri		**Nilai *cosine similarity***
q1	*q2*	Tidak ada
q1	*q4*	0
q2	*q4*	0

Dari hasil diatas menunjukkan bahwa *q1* dan *q4* tidak memiliki kesamaan sama sekali pada klausa kondisi filter. Begitu juga antara *q2* dan *q4*. Hal ini menunjukkan bahwa (*q1*,*q4*) dan (*q2*,*q4*) termasuk SQL klon tipe C.

Jika hasil dari langkah 2 sampai 4 mengasilkan kemiripan sama persis. Maka dapat dimasukkan ke dalam tipe A atau B. Sedangkan jika hanya mirip pada langkah 2 dan 3 maka dapat dimasukkan kedalam tipe C.

Untuk membedakan antara tipe A dan tipe B diperlukan langkah tambahan yaitu membandingkan hash dari kedua kueri yang memiliki hasil kemiripan persis dari langkah 2 sampai 4. Hal ini diperukan untuk mengantisipasi jika ada dua kueri yang memiliki jumlah kolom yang sama, memiliki kolom yang sama, dari tabel yang sama dan memiliki klausa filter yang identik. Tetapi tidak memiliki nama alias dan prefiks yang sama. Alias dan prefiks dihilangkan pada saat praproses sehingga akan sulit membedakan antara tipe A dan B. Oleh karena itu perlu dibandingkan hash dari kueri sebelum dilakukan praproses. Cara ini efektif untuk mendeteksi tipe A, dan penggunaan hasil hash dari kode untuk perbandingan juga pernah dilakukan pada penelitian Hotta dkk. (Hotta et al., 2014). Contoh perbandingan hash dari kueri ditunjukkan pada Tabel 3.11. berikut:

Tabel 3.11. Komparasi hash skrip SQL

Kueri	**Kueri setelah praproses**	**Aktual kueri**	**Nilai hash aktual kueri**
Q1	`select nama from mahasiswa`	`select nama from mahasiswa`	93b1fdd08180502cb4c7fe4aa309b395
Q2	`select nama from mahasiswa`	`select m.nama as mhs from mahasiswa as m`	c7a8527d7ddcf144f3954cae748df2c7

Sesuai dengan contoh diatas, jika hanya membandingkan dari hasil kueri yang telah dilakukan praproses maka tidak ada perbadaan antara tipe A dan tipe B. sehingga perlu dilakukan pengecekan hash aktual kueri. Jika nilai hash dua kueri sama maka dikatagorikan klon tipe A dan jika berbeda maka masuk tipe B.

E. Kasus SQL Klon tipe D

Untuk kasus tipe D atau klon parsial pada subkueri, akan melalui langkah yang sama. Karena klausa subkueri akan dipisahkan menjadi kueri tersendiri sebelum dilakukan peroses deteksi. Perbedaannya adalah klausa subkueri yang telah dipisahkan akan diberi label khusus. Dan jika klausa subkueri tersebut terdeteksi sebagai klon dengan kueri lain, maka dengan label tersebut klausa

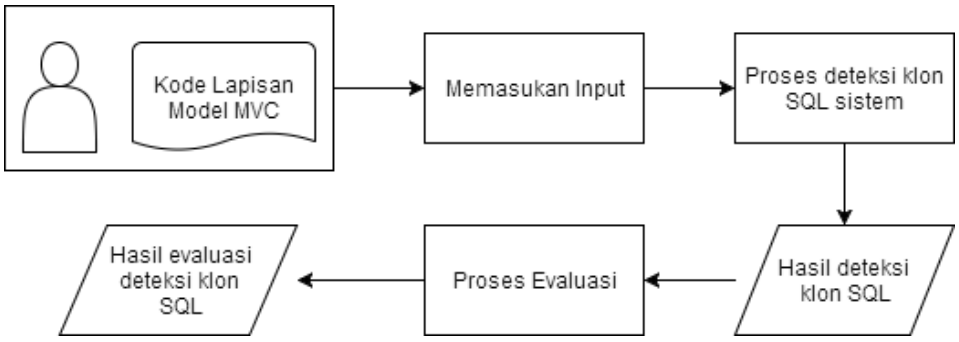
subkueri dan kueri yang dianggap sama akan dideteksi sebagai klon tipe D atau klon parsial.

3.1.3. Evaluasi Hasil

Pada penelitian ini, untuk mengevaluasi hasil deteksi klon yang dideteksi oleh sistem digunakan perhitungan akurasi, *precission*, *recall* dan *f-measure*. Evaluasi akurasi ini digunakan untuk mengetahui ketepatan sistem dalam mendeteksi klon SQL dibandingkan dengan dataset SQL skrip yang telah terlabeli klon SQL. SQL klon yang telah terlabeli akan dikonfirmasi ke pakar.

Dalam penelitian ini, satu dataset merepresentasikan satu project sistem berbasis MVC. Dan evaluasi akurasi, *precission*, *recall* dan *f-measure* dilakukan pada setiap project atau dataset.

Proses evaluasi akurasi, precision dan recall ini akan dilakukan untuk keseluruhan klon SQL dan setiap tipe klon yang didefinisikan. Gambar 3.8. berikut menunjukkan alur skenario evaluasi hasil deteksi klon SQL.



Gambar 3.8. Alur skenario evaluasi hasil deteksi

Pengguna memasukkan input ke sistem berupa kode-kode pada lapisan model. Sistem akan melakukan proses deteski klon SQL pada kode-kode inputan pengguna. Setelah proses selesai, sistem akan menampilakan hasil deteksi klon SQL. Selanjutnya akan dilakukan proses evaluasi dengan membandingkan hasil deteksi sistem dengan data aktual klon SQL pada dataset yang telah dilabeli sebelumnya. Hasil proses evaluasi adalah nilai akurasi, precision dan recall deteksi sistem terhadap data aktual klon SQL pada dataset.

3.2. Percobaan Awal

Tujuan dilakukan percobaan awal ini adalah untuk mensimulasikan langkah-langkah deteksi klon SQL di lapisan model pada kerangka kerja MVC. Pada percobaan awal ini mengangkat sebuah sistem perbankan sederhana yang mencakup cabang bank, akun bank, dan pelanggan. Pada penelitian ini yang menjadi fokus adalah lapisan model. Sehingga proses deteksi hanya dilakukan pada lapisan model. Berikut kode pada lapisan model dalam contoh kasus perbankan sederhana.

Tabel 3.12. Kode pada lapisan model

```
/**
* kelas Model_customer
*/
class Model_customer extends CI_Model{

  function __construct(){
          parent::__construct();
  }

  function customer_get($customer_id){
          $this->db->select('customer_id, customer_name, customer_street,
                      customer_city');
          if ($customer_id!=NULL){
              $this->db->where('customer_id',$customer_id); }
           $customer = $this->db->get('customer');
          return $customer->result();
  }

  function customer_balance($customer_id){
          $statement = "select c.customer_id, c.customer_name, a.balance
                        from customer c, account a
                        where c.customer_id = a.customer_id
                        and c.customer_id = ".$customer_id;
          $data = $this->db->query($statement);
          return $data->result();
  }
}

/**
* kelas Model_branch
*/
class Model_branch extends CI_Model{

function __construct(){
            parent::__ construct();
    }

function all_branch($id_branch)
    {
            $query = "select id_branch, branch_name, branch_city
                            from branch";
            if ($id_branch!=NULL) {
                    $query .=  " where id_branch = ".$id_branch;
            }
            $branch = $this->db->query($query);
    }
```

```
    function branch_account($id_branch)
    {
        $branch_account = $this->db->query("select account_number, balance
        from account, branch where branch.id_branch = account.id_branch and
        branch.id_branch = $branch");
         return $branch_account->result();
    }
}

/**
* Kelas Model_account
*/
class Model_account extends CI_Model
{
   function __construct(){
            parent::__construct();
   }
   function account_getall(){
            $account = $this->db->get('account');
            return $account->return();
   }
   function account_branchall()
   {
        $acc_custom = $this->db->query('select id_branch code, branch_name as
         branch, branch_city location from branch');
        return $acc_custom->result();
   }
   function account_balance($id_customer){
        $query = "select customer.customer_id, customer.customer_name,
        account.balance from customer , account where customer.customer_id =
        account.customer_id";
            if ($id_customer!=NULL) {
                    $query .=  " and customer.id_customer = ".$id_customer;
            }
            $balance = $this->db->query($query);
   }
}
```

Langkah pertama adalah mengekstraksi semua fungsi kelas pada lapisan model. Langkah ini ditunjukkan dengan kotak garis putus pada Tabel 3.12. informasi yang diambil pada proses ini adalah kode fungsi, nama fungsi, nama kelas model. Tahap selanjutnya adalah mengidentifikasi kueri-kueri yang mungkin pada setiap fungsi. Identifikasi kueri ini meliputi kueri yang dibangun dengan *raw* SQL dan *query builder*. Pada tahap identifikasi kueri ini juga dilakukan normalisasi variabel pada kueri. Hasil identifikasi kueri ditunjukkan pada Tabel 3.13 berikut.

Tabel 3.13. Hasil Ekstraksi Kueri Setiap Fungsi

Model	**Fungsi**	**Kueri**	**Kode**
Model_customer	__construct	Tidak ada	-
Model_customer	customer_ get	`$this->db->select('customer_id, customer_name, customer_street, customer_city');` `$this->db->get('customer');`	**Q1**

Model	Fungsi	Kueri	Kode

		`$this->db->select('customer_id, customer_name, customer_street, customer_city');`
		`$this->db->where('customer_id', "X");`
		`$this->db->get('customer');`
			Q2

Model_customer	customer_ balance	`$statement = "select c.customer_id, c.customer_name, a.balance`
		`from customer c, account a`
		`where c.customer_id = a.customer_id and c.customer_id = "."X";`
		`$this->db->query($statement);`
			Q3

Model_branch	__construct	Tidak ada	-

Model_branch	all_branch	`$query = "select id_branch, branch_name, branch_city`
		`from branch";`
		`$this->db->query($query);`
			Q4

		`$query = "select id_branch, branch_name, branch_city`
		`from branch";`
		`$query .= " where id_branch = "."X";`
		`$this->db->query($query);`
			Q5

Model_branch	branch_ account	`$this->db->query("select account_number, balance from account, branch where branch.id_branch = account.id_branch and branch.id_branch = X");`	**Q6**
Model_account	__construct	Tidak ada	
Model_account	account_ getall	`$this->db->get('account');`	**Q7**
Model_account	account_ branchall	`$this->db->query('select id_branch code, branch_name as branch, branch_city location from branch');`	**Q8**
Model_account	account_ balance	`$query = "select customer.customer_id, customer.customer_name, account.balance from customer , account where customer.customer_id = account.customer_id";` `$this->db->query($query);`	**Q9**

		`$query = "select customer.customer_id, customer.customer_name, account.balance from customer , account where customer.customer_id = account.customer_id";`
		`$query .= " and customer.id_customer = "."X";`
		`$this->db->query($query);`
			Q10

Setelah kueri-kueri pada setiap fungsi berhasil diidentifikasi, proses selanjutnya adalah translasi SQL skrip. Kueri-kueri tersebut akan ditranslasikan menjadi Bahasa SQL. Hasil dari translasi dituntukkan pada Tabel 3.14 berikut ini.

Tabel 3.14. Translasi Kueri ke skrip SQL

Model	Fungsi	SQL skrip	kode
Model_customer	customer_ getall	`Select customer_id, customer_name, customer_street, customer_city from customer`	**Q1**
Model_customer	customer_ getone	`Select customer_id, customer_name, customer_street, customer_city From customer Where customer_id = "1"`	**Q2**

Model	Fungsi	SQL skrip	kode
Model_customer	customer_ balance	select c.customer_id, c.customer_name, a.balance from customer c, account a where c.customer_id = a.customer_id and c.customer_id = "1"	**Q3**
Model_branch	all_branch	select id_branch, branch_name, branch_city from branch	**Q4**
		select id_branch, branch_name, branch_city from branch where id_branch = "1"	**Q5**
Model_branch	branch_ account	select account_number, balance from account, branch where branch.id_branch = account.id_branch and branch.id_branch = "1"	**Q6**
Model_account	account_ getall	Select * from account	**Q7**
Model_account	account_ branchall	select id_branch code, branch_name as branch, branch_city location from branch	**Q8**
Model_account	account_ balance	select customer.customer_id, customer.customer_name, account.balance from customer , account where customer.customer_id = account.customer_id	**Q9**
		select customer.customer_id, customer.customer_name, account.balance from customer , account where customer.customer_id = account.customer_id and customer.id_customer = “1”	**Q10**

Proses selanjutnya adalah melakukan praproses SQL skrip, hal ini dilakukan untuk memudahkan proses deteksi dengan menyamakan beberapa bentuk SQL. Proses praproses SQL skrip berdasarkan langkah praproses yang telah dijelaskan sebelumnya. Hasil praproses SQL skrip ditunjukkan pada Tabel 3.15. berikut.

Tabel 3.15. Hasil Praproses SQL skrip

Kode	Fungsi	SQL skrip
Q1	customer_ getall	Select customer_id, customer_city, customer_name, customer_street from customer
Q2	customer_ getone	Select customer_id, customer_city, customer_name, customer_street From customer Where customer_id = X
Q3	customer_ balance	select balance, customer_id, customer_name from customer, account where customer_id = customer_id and customer_id = X
Q4	all_branch	select branch_city, branch_name, id_branch from branch
Q5	all_branch	select branch_city, branch_name, id_branch from branch where id_branch = X

Kode	Fungsi	SQL skrip

Q6	branch_ account	`select account_number, balance`
		`from account, branch`
		`where id_branch = id_branch and id_branch = "X"`

Q7	account_ getall	`Select * from account`
Q8	account_ branchall	`select branch_city, branch_name, id_branch` `from branch`

Q9	account_ balance	`select balance, customer_id, customer_name`
		`from customer , account`
		`where customer_id = customer_id`

Q10	account_ balance	`select balance, customer_id, customer_name`
		`from customer , account`
		`where customer_id = customer_id`
		`and customer_id = X`

Proses selanjutnya adalah parsing skrip SQL. Skrip SQL akan diparsing menjadi tiga bagian utama yaitu berdasarkan kata kunci klausa SELECT, FROM, dan klausa kondisi filter (WHERE, GROUP BY, HAVING, ORDER BY). Pada saat parsing klausa SELECT, informasi yang diambil adalah jumlah kolom. Jumlah kolom tersebut akan digunakan untuk proses komparasi klausa SELECT. Tabel 3.16. berikut adalah hasil dari parsing dan pengambilan informasi jumlah kolom pada klausa SELECT.

Tabel 3.16. Jumlah kolom klausa SELECT

Kode SQL	Fungsi	Jumlah kolom		Kode SQL	Fungsi	Jumlah kolom
Q1	customer_getall	**4**		Q7	account_getall	**1**
Q2	customer_getall	**4**		Q8	account_branchall	**3**
Q3	customer_balance	**3**		**Q9**	account_balance	**3**
Q4	all_branch	**3**		**Q10**	account_balance	**3**
Q5	all_branch	**3**				
Q6	branch_account	**2**				

Dari hasil tersebut, kemudian akan dilakukan komparasi jumlah kolom. Jika jumlah kolom tidak sama maka kueri tersebut dinyatakan bukan klon. Syarat kedua adalah kueri yang dikomparasi bukan berasal dari fungsi yang sama walaupun memiliki jumlah kolom yang sama. Dalam hal ini, (Q1,Q2), (Q4,Q5) dan (Q10,Q11) tidak dimasukkan sebagai pasangan SQL yang dikomparasi walaupun memiliki jumlah kolom yang sama. Hanya Q1 dan Q2 yang memiliki jumlah kolom

4 tetapi Q1 dan Q2 berada pada fungsi yang sama, maka dianggap tidak ada klon untuk Q1 dan Q2. Sedangkan Q6 dan Q7 tidak memiliki jumlah kolom yang sama sehingga dipastikan tidak akan ditemukan klon. Selanjutnya akan dilakukan komparasi nama kolom pada SELECT yang memiliki jumlah kolom yang sama seperti ditunjukkan pada Tabel 3.17. berikut ini.

Tabel 3.17. Daftar Nama Kolom klausa SELECT

Kode SQL	**Fungsi**	**Jumlah Kolom**	**Kolom**
Q3	customer_balance	3	`balance, customer_id, customer_name`
Q4	all_branch	3	`branch_city, branch_name, id_branch`
Q5	all_branch	3	`branch_city, branch_name, id_branch`
Q8	account_branchall	3	`branch_city, branch_name, id_branch`
Q9	account_balance	3	`balance, customer_id, customer_name`
Q10	account_balance	3	`balance, customer_id, customer_name`

Dari informasi jumlah kolom dan nama kolom pada klausa SELECT. Maka ditemukan pasangan kueri yang dapat memiliki klausa SELECT yang sama. Hasil pasangan kueri yang sesuai dengan komparasi klausa SELECT ditunjukkan pada Tabel 3.18. berikut.

Tabel 3.18. Hasil Akhir Komparasi Klausa SELECT

Jumlah kolom	**Kolom**	**Pasangan komparasi**
3	`balance, customer_id, customer_name`	(Q9,Q3) (Q10,Q3)
3	`branch_city, branch_name, id_branch`	(Q4,Q8) (Q5,Q8)

Langkah selanjutnya adalah komparasi klausa FROM. Dari hasil komparasi klausa SELECT yang menghasilkan beberapa pasangan komparasi SQL akan di komparasi klausa FROM setiap pasangan tersebut. Pada klausa FROM minimal tabel yang sama adalah satu tabel, jika jumlah dan nama tabel sama maka akan dilanjutkan pada proses selanjutnya. Jika tidak sama semua antara jumlah dan nama klom, tetapi memiliki minimal satu tabel yang sama maka dideteksi sebagai klon tipe C. Tabel 3.19. berikut ini menunjukkan proses komparasi pada klausa FROM.

Tabel 3.19. Komparasi klausa FROM

Pasangan Komparasi	**Jumlah Tabel**	**Tabel**
(Q9,Q3)	Q9 = 2 Q3 = 2	Q9 {customer , account} Q3 {customer , account}
(Q10,Q3)	Q10 = 2 Q3 = 2	Q10 {customer , account} Q3 {customer , account}
(Q4,Q8)	Q4 = 1 Q8 = 1	Q4 {branch} Q8 {branch}
(Q5,Q8)	Q5 = 1 Q8 = 1	Q5 {branch} Q8 {branch}

Dari hasil di atas menunjukkan semua pasangan komparasi memiliki jumlah tabel dan nama tabel yang sama. Sehingga akan dilakukan proses selanjunya yaitu komparasi klausa filter. Pada komparasi klausa filter digunakan metode *cosine similarity*. Sebelum melakukan perhitungan *cosine similarity*, terlebih dahulu dihitung TF (*term frequency*) setiap kata pada klausa filter dan kemudian menghitung similaritas dengan *cosine similarity* setiap pasangan komparasi.

Contoh perhitungan *cosine similarity* klausa filter pada kueri Q3 dan Q9 sebagai berikut:

Q3	`where customer_id = customer_id` `and customer_id = X`
Q9	`where customer_id = customer_id`

Kemudian menghitung setiap term pada kedua klausa tersebut. Perhitungan term pada kedua klausa tersebut dapat ditunjukkan sebagai berikut.

term	**where**	**customer_id**	**=**	**and**	**x**
Q3	1	3	2	1	1
Q9	1	2	1	0	0

Kemudian menghitung similaritas dengan *cosine similarity*.

$$\text{Sim}\,(Q3, Q9) = \frac{(1*1)+(3*2)+(2*1)+(1*0)+(1*0)}{\sqrt{1^2+3^2+2^2+1^2+1^2} * \sqrt{1^2+2^2+1^2+0+0}} = \mathbf{0.92}$$

Hasil komparasi klausa filter ditunjukkan pada Tabel 3.20. berikut.

Tabel 3.20. Hasil Cosine similarity pada klausa filter

Pasangan komparasi		**Nilai *cosine similarity***
Q9	*Q3*	0.92
Q10	*Q3*	1
Q4	*Q8*	Tidak ada klausa filter
Q5	*Q8*	0

Dari hasil pada tabel 3.20. diketahui bahwa pasangan (Q9,Q3) dan (Q5,Q8) merupakan klon tipe C. karena klausa filter pada kedua pasangan SQL tersebut tidak sama. Pada kasus (Q5,Q8) yang memiliki nilai 0 dikarenakan pada Q5 tidak memiliki klausa filter tetapi Q8 memiliki klausa filter, maka termasuk klon tipe C. Pasangan (Q10,Q3) memiliki nilai cosine 1 atau dalam kata lain sama persis. Maka dapat disimpulkan bahwa (Q10,Q3) merupakan SQL klon tipe A karena dari komparasi klausa SELECT sampai kondisi filter memiliki hasil yang sama. Pasangan (Q4,Q8) merupakan klon tipe B. Karena pada Q8 ada penambahan alias yang dihilangkan pada proses praproses SQL skrip. Tabel 3.21. berikut adalah hasil akhir dari deteksi klon SQL pada lapisan model.

Tabel 3.21. Hasil Deteksi Klon SQL

Pasangan SQL Klon		**Tipe Klon SQL**
Q9	*Q3*	Tipe C
Q10	*Q3*	Tipe A
Q4	*Q8*	Tipe B
Q5	*Q8*	Tipe C

BAB 4
HASIL DAN PEMBAHASAN

4.1. Pengumpulan Dataset

Penelitian ini membutuhkan dataset yang digunakan untuk uji coba pada pendekatan yang diusulkan. Dataset yang digunakan adalah kode sumber dari beberapa aplikasi berbasis web MVC, khususnya yang menggunakan kerangka kerja CodeIgniter. Kode sumber aplikasi yang digunakan pada penelitian ini adalah kode sumber pada lapisan model. Tabel 4.1 berikut merupakan daftar dari aplikasi yang digunakan untuk uji coba dalam penelitian ini.

Tabel 4.1. Daftar Kode Sumber

No.	Nama Aplikasi	LOC	Jumlah Kelas lapisan Model	LOC Lapisan Model	Bahasa / Kerangka kerja
1	ERPKokin	32 KLOC	9 kelas	928 LOC	PHP/CodeIgniter
2	Elogbook	30 KLOC	10 kelas	686 LOC	PHP/CodeIgniter
3	PonpeskuApp	38 KLOC	13 kelas	1551 LOC	PHP/CodeIgniter
4	Akutansi	41 KLOC	25 kelas	1889 LOC	PHP/CodeIgniter
5	WS PD Dikti	930 LOC	12 kelas	930 LOC	SQL
6	Mutasi	38 KLOC	10 kelas	992 LOC	PHP/CodeIgniter

Aplikasi yang digunakan sebagai dataset merupakan aplikasi hasil penelitian sebelumnya dan aplikasi riil. Aplikasi ERPkokin (Yudoyo, 2015) merupakan aplikasi ERP (*Enterprise Resource Planning*) pada suatu perusahaan. Aplikasi PonpeskuApp (Akbar, 2014) merupakan aplikasi manajeman untuk institusi pendidikan. Aplikasi Elogbook merupakan aplikasi untuk menyimpan semua kegiatan atau aktivitas keseharian pegawai dan penilaian berdasarkan waktu per aktivitas yang dikonfirmasi oleh atasan per masing-masing pegawai. Aplikasi Akutansi merupakan aplikasi pencatatan sistem keuangan rumah sakit. Aplikasi Mutasi merupakan aplikasi yang telah dimutasi kode pada lapisan model untuk mengakomodasi variasi-variasi kueri, aplikasi mutasi adalah mutasi dari aplikasi Pajakonline (Zulkarnaein, 2014) yang merupakan aplikasi yang digunakan untuk pelaporan pajak ke pemerintah. Lima aplikasi tersebut merupakan aplikasi berbasis MVC dengan menggunakan kerangka kerja MVC CodeIgniter. Sedangkan untuk

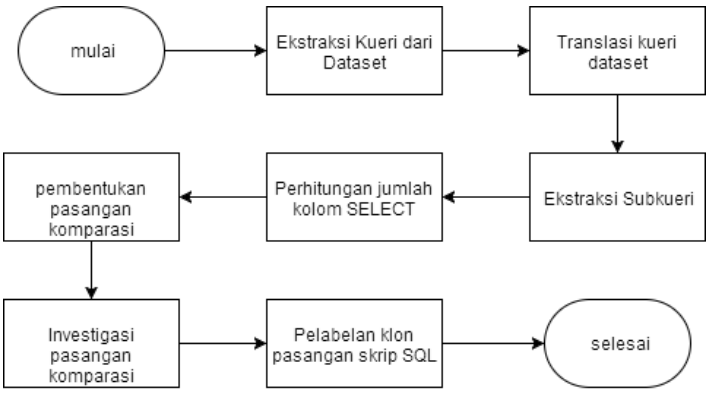
aplikasi Web Service (WS) Pangkalan Data (PD) Dikti merupakan dataset dalam bentuk skrip SQL.

Enam aplikasi yang dijadikan objek uji coba memiliki karakteristik berbeda-beda. Aplikasi ERPkokin, Ponpesku, dan Mutasi adalah aplikasi yang sebagian besar kuerinya menggunakan bentuk raw SQL. Aplikasi Elogbook dan Akutansi adalah aplikasi yang sebagian besar kuerinya menggunakan bentuk *query builder*. Sedangkan kueri pada aplikasi WS PD Dikti sudah dalam bentuk SQL skrip.

4.1.1. Pelabelan Dataset

Pada penelitian di area kode klon, ada dua bentuk atau model dataset yang sering digunakan untuk mengevaluasi metode deteksi klon. Pertama adalah dataset yang sudah terlabeli klon beserta tipe-tipenya dan yang kedua adalah dataset yang dibentuk dari proses mutasi kode. Dataset terlabeli yang banyak digunakan untuk evaluasi kode klon adalah dataset yang dibuat oleh Bellon (Bellon, Koschke, Antoniol, Krinke, & Merlo, 2007). Bellon melakukan pelabelan manual dari hasil beberapa kode klon detektor. Hasil pelabelan manual tersebut dijadikan sebagai acuan kebenaran. Dalam penelitian ini, pelabelan manual dilakukan untuk membentuk dataset sebagai acuan kebenaran.

Pada penelitian ini, dataset yang dikumpulkan akan dilakukan pelabelan klon beserta tipe-tipenya. Proses pelabelan ini dilakukan dengan beberapa langkah. Pelabelan klon mengacu pada definisi klon dan tipe-tipe klon yang telah didefinisikan sebelumnya. Setiap langkah dari proses pelabelan dilakukan secara manual. Adapun langkah-langkah pelabelan ditunjukkan pada Gambar 4.1 sebagai berikut:



Gambar 4.1. Langkah pelabelan dataset

1. **Ekstraksi Kueri dari Dataset**

   Ekstraksi kueri digunakan untuk menginvestigasi kueri-kueri yang ada pada suatu fungsi di kelas-kelas lapisan model. Contoh ekstraksi kueri pada suatu fungsi sebagai berikut:

```
/*APLIKASI ERPKOKIN. KELAS MODEL: mpenjualan.php
function ajUpProduk($idpemesanan_produk,$isi)
{
    $statement4 = 'select har.idharga_produk, har.harga_produk from harga_produk har, produk pro where har.idproduk = pro.idproduk and har.tgl_akhir_berlaku = (select max(tgl_akhir_berlaku) from harga_produk where idproduk  = har.idproduk) and pro.idproduk = ?';
    $qry = $this->db->query($statement4,$isi);
    $har = $qry->result();

    $statement = 'update pemesanan_produk set idproduk = ?, tgl_pemesanan = curdate(), idharga_produk = ? where idpemesanan_produk = ?';
    $this->db->query($statement,array($isi,$har[0]->idharga_produk, $idpemesanan_produk));

    $statement2 = 'select nama_produk from produk where idproduk = ?';
    $query = $this->db->query($statement2,$isi);
    $nama = $query->result();

    $statement3 = 'update log_pemesanan_produk set produk = ?, tgl_pemesanan = curdate(), harga = ? where idpemesanan = ?';
    $this->db->query($statement3,array($nama[0]->nama_produk,$har[0]->harga_produk, $idpemesanan_produk));
}
```

   Kueri yang diekstraksi hanya kueri SELECT *statement*. Kueri yang lain seperti UPDATE dan DELETE diabaikan.

2. **Translasi Dataset**

   Langkah ini digunakan untuk mengubah kueri SELECT dari bentuk bahasa PHP ke skrip SQL. Pada langkah ini juga dilakukan penyamaan semua bentuk value atau literal pada skrip SQL. Dengan penggantian value ini, semua value atau literal akan diintrepretasi menjadi bentuk yang sama. Contoh:

```
$statement2 = 'select nama_produk from produk where idproduk = ?';
$query = $this->db->query($statement2,$isi);
```

Menjadi

```
select nama_produk from produk where idproduk = 1
```

3. Ekstraksi subkueri

Pada beberapa kasus, akan ditemukan skrip SQL yang memiliki subkueri atau penggabungan kueri dengan UNION *statement*. Setiap kueri tersebut dilakukan ekstraksi setiap sub-kuerinya. Contoh:

```
select har.idharga_produk, har.harga_produk
from harga_produk har, produk pro
where har.idproduk = pro.idproduk and har.tgl_akhir_berlaku = (
   select max(tgl_akhir_berlaku) from harga_produk where idproduk  =
   har.idproduk
) and pro.idproduk = 1
```

Hasil ekstraksi subkuerinya adalah:

```
   select max(tgl_akhir_berlaku) from harga_produk where idproduk  =
   har.idproduk
```

Sehingga ada dua kueri yang terbentuk yaitu kueri induk dan subkuerinya.

4. Perhitungan jumlah kolom

Langkah selajutnya adalah menghitung jumlah kolom pada klausa SELECT. Perhitungan jumlah kolom ini dimaksudkan untuk memudahkan pada saat pembentukan pasangan komparasi. Selain itu, jumlah kolom yang sama merupakan salah satu syarat sepasang skrip SQL dikategorikan sebagai klon.

5. Pembentukan pasangan komparasi

Dari hasil perhitungan jumlah kolom pada langkah sebelumya akan dibuat pasangan komparasi skrip SQL. Syarat pasangan komparasi untuk skrip SQL adalah memiliki jumlah kolom yang sama dan dari fungsi yang berbeda. Skrip SQL dari fungsi yang sama tidak dilakukan komparasi. Pada proses pembentukan pasangan komparasi, hasil ekstraksi subkueri dianggap menjadi kueri independen.

6. **Investigasi pasangan komparasi sesuai dengan definisi tipe-tipe klon**
   Setelah pasangan komparasi skrip SQL dibentuk, selanjutnya dilakukan investigasi pasangan komparasi sesuai dengan definisi tipe-tipe klon yang telah didefinisikan sebelumnya.
7. **Pelabelan klon pasangan skrip SQL**
   Jika ditemukan klon pada pasangan komparasi. Pasangan kueri tersebut akan diberi label klon dan informasi tipe klonnya.

4.2. Skenario Pengujian

Pada penelitian ini, untuk mengevaluasi hasil deteksi klon yang dideteksi oleh sistem digunakan perhitungan akurasi, precision dan recall. Evaluasi ini digunakan untuk mengetahui ketepatan sistem dalam mendeteksi klon SQL dibandingkan dengan dataset skrip SQL yang telah terlabeli klon SQL. Klon SQL yang telah terlabeli telah dikonfirmasi oleh pakar.

Dalam penelitian ini, satu dataset merepresentasikan satu project sistem berbasis MVC. Dan evaluasi akurasi dilakukan pada setiap project atau dataset.

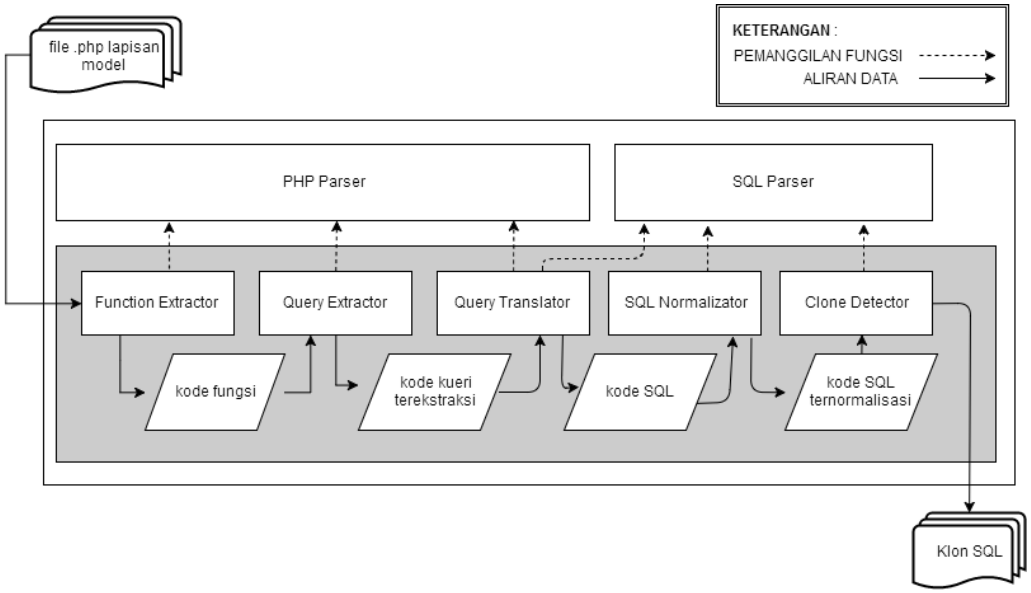
Pengguna memasukkan input ke sistem berupa kode-kode pada lapisan model. Sistem akan melakukan proses deteski klon SQL pada kode-kode inputan pengguna. Setelah proses selesai, sistem akan menampilkan hasil deteksi klon SQL. Selanjutnya akan dilakukan proses evaluasi dengan membandingkan hasil deteksi sistem dengan data aktual klon SQL pada dataset yang telah dilabeli sebelumnya. Hasil proses evaluasi adalah nilai akurasi deteksi sistem terhadap data aktual klon SQL pada dataset.

4.3. Implementasi

Dalam penelitian ini implementasi dan uji coba akan dibangun dalam lingkungan pengembangan sebagai berikut:

- Sistem Operasi : Windows 10 64 bit
- RAM : 4 GB
- Processor : Intel Core i5
- IDE : Sublime Text 3
- *Software* Pendukung : XAMPP, Chrome browser

Gambar 4.2 merupakan arsitektur sistem dari proses implementasi. Input dari sistem adalah kode sumber pada lapisan model dan output dari sistem adalah informasi klon SQL yang meliputi tipe klon dan pasangan klon SQL (*clone pair*). Pada penelitian ini menggunakan dua pustaka utama yaitu PHP parser (Popov, 2017) dan SQL parser (Swanhart, 2017). PHP parser digunakan untuk memparsing kode sumber di lapisan model, sedangkan SQL parser digunakan untuk memparsing kode SQL yang berhasil diekstraksi untuk proses deteksi. Sistem yang dibangun dibagi menjadi lima bagian utama sesuai dengan metodologi pada penelitian ini. *Function Extractor* merupakan bagian dari sistem yang bertugas untuk mengekstraksi setiap fungsi pada kelas-kelas lapisan model. *Query Extractor* berfungsi untuk mengekstrasi kemungkinan kueri yang ada pada suatu fungsi. *Function Extractor* dan *Query Extractor* menggunakan PHP Parser sebagai pemecah kode PHP. *Query Translator* berfungsi untuk mentransalasikan kueri yang berhasil diekstraksi menjadi SQL skrip. *Query Translator* menggunakan PHP parser dan SQL parser. SQL Normalizator berfungsi untuk melakukan pra-proses pada skrip SQL. Terakhir adalah *Clone Detector* yang berfungsi sebagai pendeteksi klon SQL.



Gambar 4.2. Arsitektur sistem

Berikut ini dijelaskan terkait pseudocode proses deteksi dari implementasi sistem. Gambar 4.3. merupakan pseudocode dari proses deteksi klon SQL. Gambar 4.4., Gambar 4.5., Gambar 4.6. berturut-turut merupakan pseudocode dari proses

evaluasi klausa SELECT, klausa FROM dan klausa filter. Pada Gambar 4.6, Gambar 4.8, Gambar 4.9 berturut-turut merupakan pseudocode dari evaluasi klon tipe A, tipe B, dan tipe C.

```
*********************************************
function: sql_clone_detection
input: pair SQL script (SQL script1,SQL script2)
output: clone type
*********************************************
parsed1 ← parsed SQL script1
parsed2 ← parsed SQL script2

select_eval ← CALL select_evaluation with parsed1, parsed2

IF select_eval = 0 THEN
RETURN not clone
ENDIF

from_eval ← CALL from_evaluation with parsed1, parsed2

IF from_eval = 0 THEN
RETURN not clone
ENDIF

filter_eval ← CALL filter_evaluation with parsed1, parsed2

IF (CALL check_CloneTypeA with select_eval, from_eval, filter_eval = true)
THEN
       IF pair is subquery THEN
       CLONE_TYPE ← 'TYPE D'
       ELSE CLONE_TYPE ← 'TYPE A'
       ENDIF
ELSE IF (CALL  check_CloneTypeB(select_eval, from_eval, filter_eval) = true)
THEN
       IF pair is subquery THEN
       CLONE_TYPE ← 'TYPE D'
       ELSE CLONE_TYPE ← 'TYPE B'
       ENDIF
ELSE IF (CALL  check_CloneTypeC(select_eval, from_eval, filter_eval) = true)
THEN
       IF pair is subquery THEN
       CLONE_TYPE ← 'TYPE D'
       ELSE CLONE_TYPE ← 'TYPE C'
       ENDIF
ENDIF
```

Gambar 4.3. Pseudocode proses deteksi klon SQL

```
*********************************************
function: select_evaluation
input: parsed pair SQL script (parsed1,parsed2)
output: status comparison
       0 = tidak sama.
       1 = sama persis, identik, memiliki kolom dan urutan yang sama.
       2 = sama, tidak identik, memiliki kolom sama tetapi urutan yang
berbeda (kandidiat type B)
*********************************************
```

```
select1 ← parse parsed1 column ref select clause
select2 ← parse parsed2 column ref select clause

IF (select1 = select2) THEN
	RETURN 1
ELSEIF (select1 and select2 have same column ref, but different sequence)
THEN
	RETURN 2
ELSE
	RETURN 3 ENDIF
```

Gambar 4.4. Pseudocode evaluasi klausa SELECT

```
*********************************************
function: from_evaluation
input: parsed pair SQL script (parsed1,parsed2)
output: status comparison
	0 = tidak sama.
	1 = sama persis, identik, memiliki kolom dan urutan yang sama.
	2 = dari tabel yang sama, tetapi urutan yang sama tidak sama.
	3 = masuk dalam syarat minimal 1 tabel yang sama
*********************************************
table_ref1 ← parse parsed1 table ref from clause
table_ref2 ← parse parsed2 table ref from clause

IF (table_ref1 = table_ref2) THEN
	IF (have same join type)
	THEN
		RETURN 1
	ELSE
		RETURN 3 ENDIF
ELSEIF (table_ref1 and table_ref2 have same table ref, but different squence)
THEN
	IF (have same join type)
	THEN
		RETURN 2
	ELSE
		RETURN 3 ENDIF
ELSEIF (table_ref1 and table_ref2 minimal have same table ref) THEN
	RETURN 3
ELSE
	RETURN 0 END IF
```

Gambar 4.5. Pseudocode evaluasi klausa FROM

```
*********************************************
function: filter_evaluation
input: parsed pair SQL script (parsed1,parsed2)
output: status comparison
		9 = tidak ada klausa filter kedua pair sql
		0 = tidak sama sama sekali.
		n>0 - n<=1 = hasil nilai cosine.
*********************************************
filter1 ← parse parsed1 filter clause
filter2 ← parse parsed2 filter clause

IF (no filter clause) THEN
	RETURN 9
ELSE IF (only one have filter clause) THEN
	RETURN 0 ENDIF
similarity <- calculate cosin similarity of filter1 and filter2;
RETURN similarity
```

Gambar 4.6. Pseudocode evaluasi klausa filter

```
*********************************************
function: check_CloneTypeA
input: select_eval,from_eval,filter_eval, parsed pair SQL script
output: status comparison
*********************************************
TypeA <- false;

IF (select_eval = 1 AND from_eval = 1 AND (filter_eval=1 OR filter_eval=9))
THEN
       IF (there is prefix) THEN
              IF ( hash(SQL script1) = hash(SQL script2) ) THEN
              TypeA <- true
       ELSE
              TypeA <- true
       ENDIF
ENDIF
RETURN TypeA
```

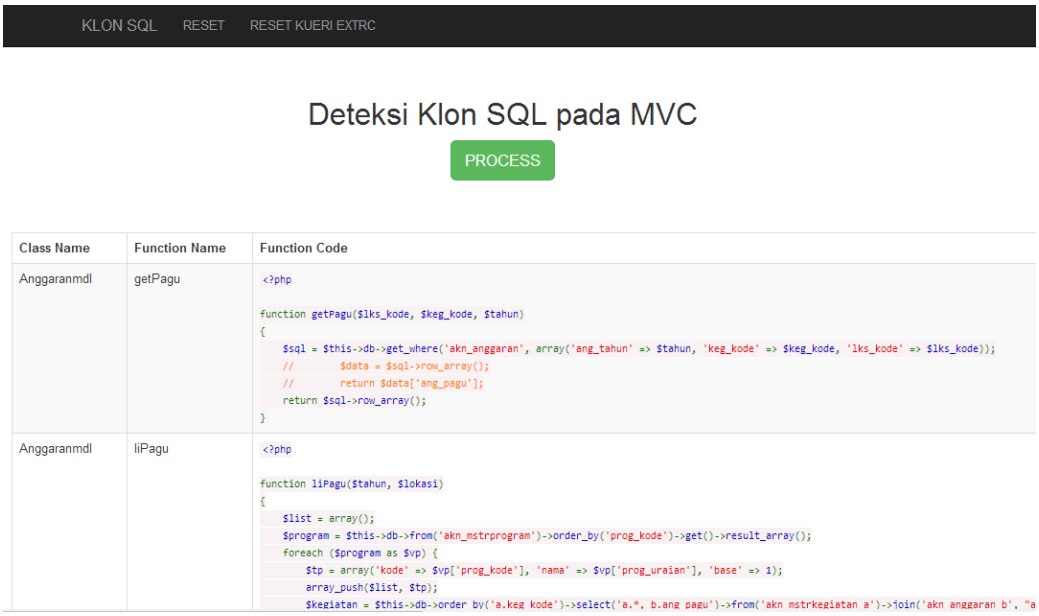
Gambar 4.7. Pseudocode evaluasi klon tipe A

```
*********************************************
function: check_CloneTypeB
input: select_eval,from_eval,filter_eval, parsed pair SQL script
output: status comparison
*********************************************
TypeB ← false;

IF (select_eval=1 AND from_eval =1 AND (filter_eval=1 OR filter_eval=9)) THEN
       IF ( there is prefix ) THEN
              IF (hash(SQL script1) != hash(SQL script2)) THEN
                     TypeB <- true
              ENDIF
        ENDIF
ENDIF
IF (select_eval=1 AND from_eval = 2 AND (filter_eval=1 OR filter_eval=9))
THEN
       TypeB ← true
ENDIF
IF (select_eval=2  AND (from_eval = 1 OR from_eval = 2) AND (filter_eval=1 OR
filter_eval=9)) THEN
       TypeB ← true
ENDIF
RETURN TypeB
```

Gambar 4.8. Pseudocode evaluasi klon tipe B

```
*********************************************
function: check_CloneTypeC
input: select_eval,from_eval,filter_eval, parsed pair SQL script
output: status comparison
*********************************************
TypeC ← false;

IF ((select_eval = 1 OR select_eval = 2) AND from_eval = 3) THEN
       TypeC ← true;
ELSEIF((select_eval = 1 OR select_eval = 2) AND (from_eval = 1 OR from_eval
=2) AND ( filter>=0 AND filter<1 )) THEN
       TypeC ← true;
RETURN TypeC
```

Gambar 4.9. Pseudocode evaluasi klon tipe C



Gambar 4.10. Tampilan sistem

Gambar 4.10 menunjukkan tampilan sistem pada proses ekstraksi fungsi dari kelas-kelas lapisan model.

Gambar 4.11. Tampilan hasil deteksi sistem

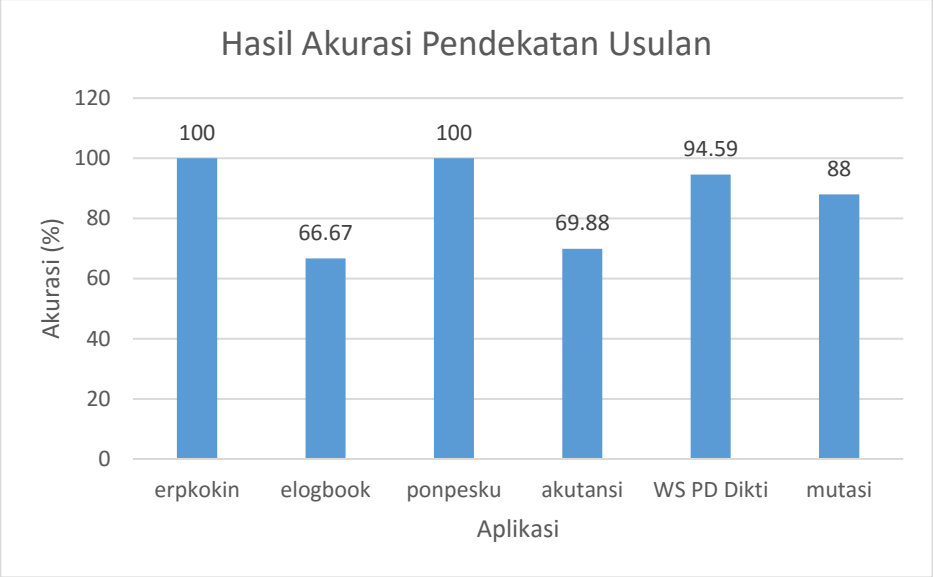
Gambar 4.11 menunjukkan tampilan sistem pada proses deteksi klon SQL pada lapisan model aplikasi berbasis MVC. Hasil dari proses deteksi adalah laporan terkait jumlah klon SQL dan persebaran tipe-tipe klon SQL.

4.4. Analisis Hasil

Analisa hasil penelitian ini mencakup pembahasan hasil evaluasi akurasi, precision dan recall sistem, pembahasan persebaran klon, dan persebaran tipe-tipe klon. Selain itu akan dibahas terkait limitasi pendekatan yang disusulkan.

4.4.1. Analisis Akurasi, Precision dan Recall Metode Deteksi

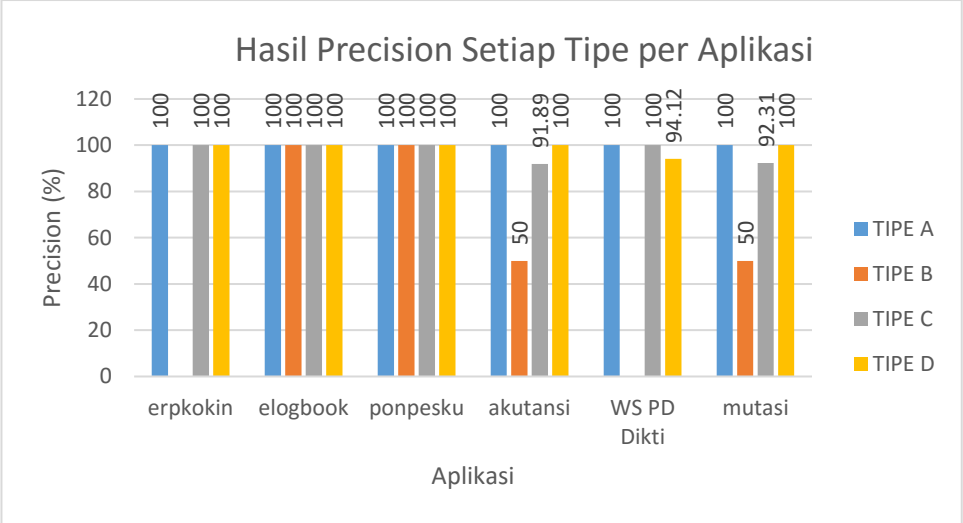
Evaluasi akurasi didapat dari membandingkan hasil keluaran sistem dengan dataset yang telah dilabeli sebelumnya. Dataset yang terlabeli menjadi acuan kebenaran dari hasil keluaran sistem. Gambar 4.12 berikut menunjukkan hasil akurasi deteksi klon SQL oleh sistem.



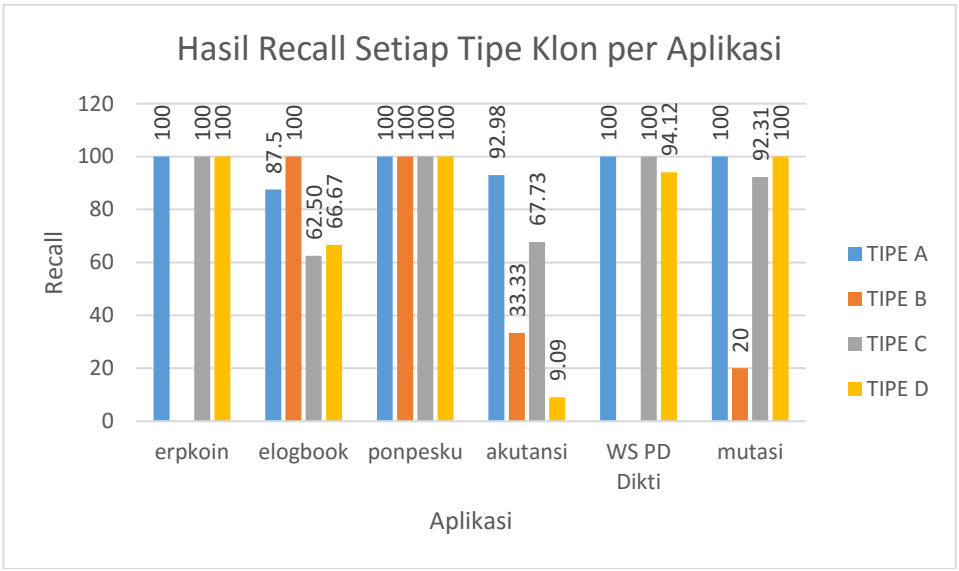
Gambar 4.12. Hasil evaluasi akurasi metode deteksi

Dari hasil evaluasi akurasi pada Gambar 4.12. menunjukkan, dua aplikasi yaitu Erpkokin dan Ponpesku menunjukkan hasil yang signifikan. Semua aktual klon terdeteksi pada dua dataset tersebut. Sedangkan pada aplikasi Elogbook sebesar 66,67%, Aplikasi Akutansi sebesar 69,88%, Aplikiasi Web Service (WS) PD Dikti sebesar 94,59%, dan Aplikasi mutasi sebesar 88%.

Gambar 4.13. menunjukkan hasil nilai precision setiap tipe-tipe klon pada setiap aplikasi. Pada aplikasi erpkokin dan WS PD Dikti tidak ada klon tipe B sehingga nilai precision tidak ada. Pada aplikasi elogbook dan ponpesku nilai precision setiap tipe sangat signifikan.



Gambar 4.13. Hasil Evaluasi Precision Setiap Tipe per Aplikasi

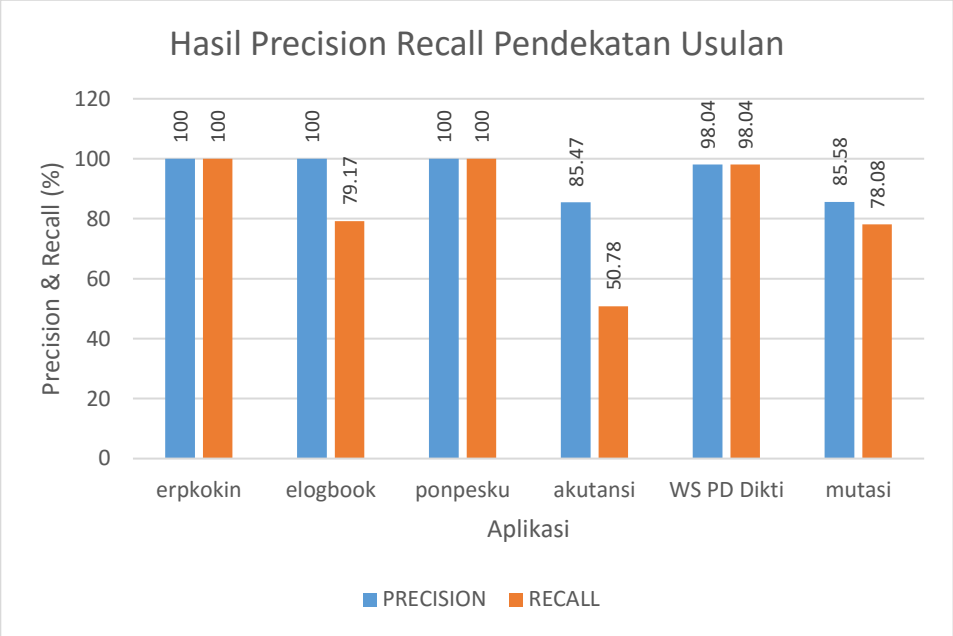


Gambar 4.14. Hasil Evaluasi Recall Setiap Tipe per Aplikasi

Gambar 4.14. menunjukkan hasil nilai recall setiap tipe-tipe klon pada setiap aplikasi. Pada aplikasi erpkokin dan WS PD Dikti tidak ada klon tipe B sehingga nilai precision tidak ada. Pada aplikasi ponpesku nilai precision setiap tipe sangat signifikan.

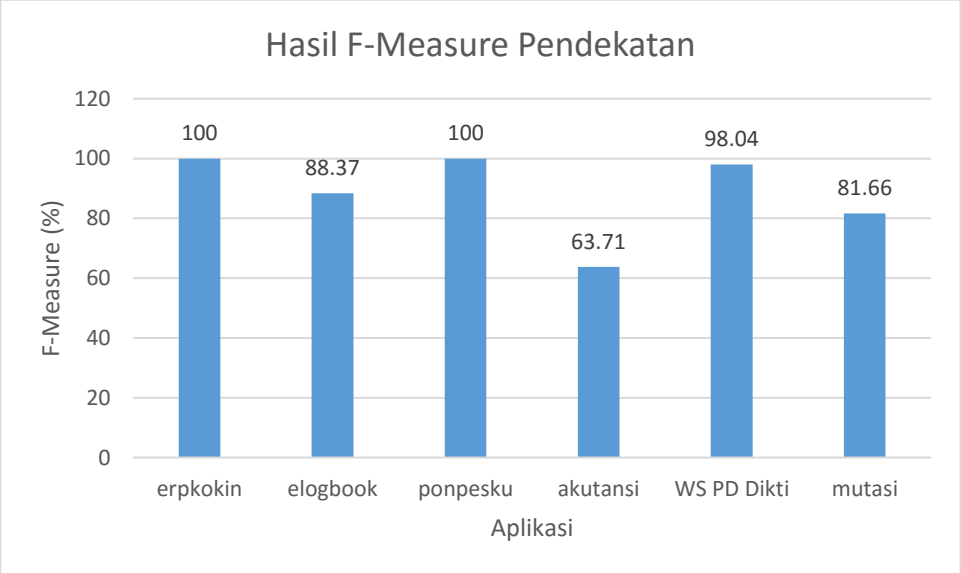
Dari hasil evaluasi precision dan recall setiap aplikasi pada Gambar 4.15 menunjukkan, dua aplikasi yaitu Erpkokin dan Ponpesku menunjukkan hasil precision dan recall yang signifikan. Sedangkan pada aplikasi Elogbook nilai precision dan recall berturut-turut 100% dan 79,17%. Pada aplikasi Akutansi nilai precision dan recall berturut-turut 85,47% dan 50,78%. Pada aplikasi WS PD Dikti

nilai precision dan recall berturut-turut 98,04% dan 98,04%. Sedangkan pada aplikasi WS PD Dikti nilai precision dan recall berturut-turut 98,04% dan 98,04%.



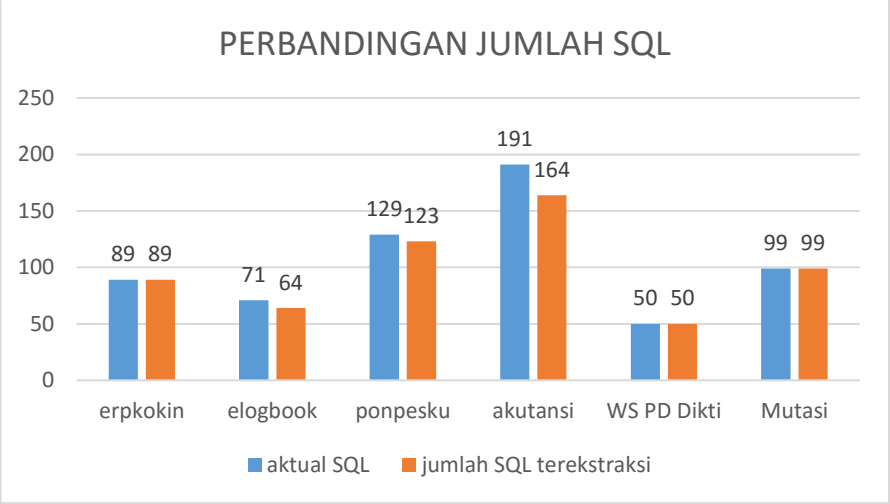
Gambar 4.15. Hasil evaluasi precision dan recall metode deteksi

Pada Gambar 4.16 menunjukkan bahwa nilai f-measure dari dua aplikasi yaitu erpkokin dan ponpesku memiliki nilai yang signifikan. Sedangkan aplikasi yang memiliki nilai f-measure terendah adalah aplikasi Akutansi.



Gambar 4.16. Hasil evaluasi *f-measure* metode deteksi

Selain itu, jumlah SQL skrip yang berhasil diekstraksi dan ditranslasi sebelum proses deteksi ditunjukkan pada Gambar 4.17. dan Tabel 4.2. berikut. Jumlah SQL skrip ini hanya mencakup jumlah SELECT *statement*.



Gambar 4.17. Perbandingan jumlah aktual skrip SQL dengan hasil ekstraksi

Tabel 4.2. Perbandingan jumlah aktual skrip SQL dengan hasil ekstraksi

Dataset	Aktual SQL skrip	Hasil ekstraksi SQL skrip sistem
Erpkokin	89	89
Elogbook	71	64
Ponpesku	129	123
Akutansi	191	164
WS PD Dikti	50	50
Mutasi	99	99

Hasil perbandingan jumlah aktual skrip SQL dengan hasil ekstraksi yang dilakukan oleh sistem menunjukkan hasil yang tidak selalu sama. Pada aplikasi Ponpesku, Elogbokk dan Akutansi jumlah hasil ekstraksi skrip SQL lebih sedikit dibandingkan dengan jumlah aktual skrip SQL dataset. Hal ini menunjukkan ada SQL skrip yang gagal diekstraksi oleh sistem.

4.4.2. Pembahasan Hasil Uji Coba

Pembahasan dilakukan untuk analisa lebih lanjut terkait dengan hasil uji coba yang sudah dilakukan sebelumnya. Pembahasan mencakup persebaran klon, persebaran tipe-tipe klon dan permasalahan yang dihadapi ketika gagal mendeteksi klon SQL.

4.4.2.1. Persebaran Klon SQL

Pada penelitan ini, komparasi skrip SQL dilakukan dengan membandingkan skrip SQL pada satu fungsi dengan SQL skrip pada fungsi yang lain. SQL skrip pada fungsi yang sama tidak dilakukan komparasi. Seperti ditunjukkan pada contoh pada berikut.

```
function get_kas_awal($key=null){
       if ($key) {
              $stm="SELECT * from kas_awal WHERE k_kas=?";
              return $this->db->query($stm,array($key))->result();
       }
       else return $this->db->get('kas_awal')->result();
}
```

Potongan kode di atas adalah bagian dari aplikasi Ponpesku. Potongan kode tersebut merupakan satu fungsi yang memiliki dua kueri atau skrip SQL yaitu:

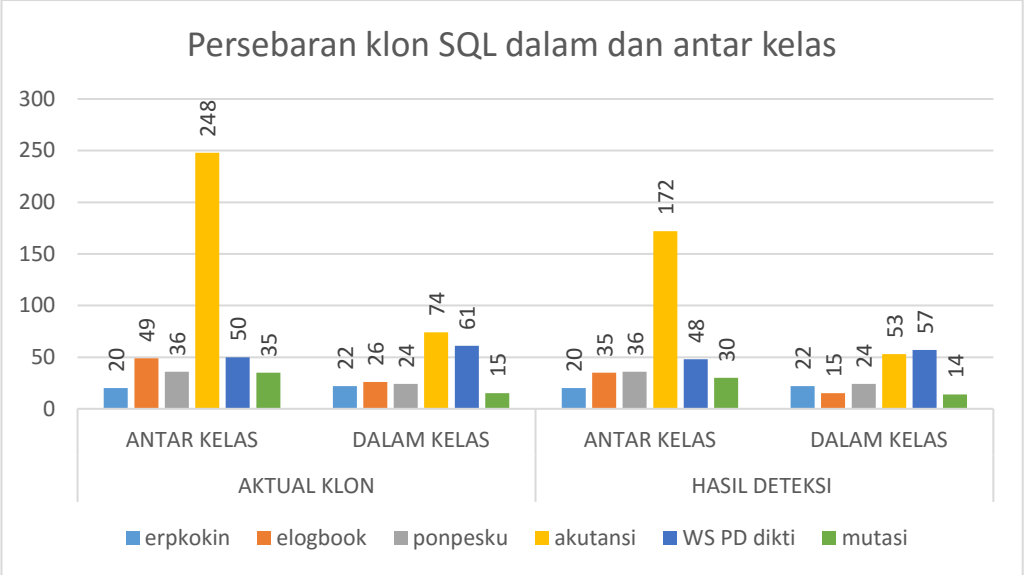
- `Q1 = SELECT * FROM kas_awal WHERE k_kas=?`
- `Q2 = SELECT * FROM kas_awal`

Skrip SQL Q1 dan Q2 tersebut tidak dilakukan komparasi karena masih dalam satu fungsi. Komparasi dilakukan dengan skrip SQL dari fungsi lain.

Klon SQL yang terdeteksi dapat dikelompokkan menjadi dua kelompok yaitu klon yang terdeteksi pada kelas yang sama dan klon yang terdeteksi antar kelas. Klon pada kelas yang sama merupakan pasangan klon yang dideteksi pada kelas yang sama tetapi dari fungsi yang berbeda. Sedangkan klon antar kelas merupakan pasangan klon yang dideteksi pada kelas yang berbeda. Persebaran hasil deteksi klon dalam suatu kelas dan antar kelas ditunjukkan pada Tabel 4.3.

Tabel 4.3. Persebaran klon SQL dalam dan antar kelas

Dataset	Klon SQL			Hasil deteksi metode		
	Antar kelas	Dalam kelas	total	Antar kelas	Dalam kelas	total
erpkokin	20	22	42	20	22	42
Elogbook	49	26	75	35	15	50
ponpesku	36	24	60	36	24	60
Akutansi	248	74	322	172	53	225
WS PD Dikti	50	61	111	48	57	105
Mutasi	35	15	50	30	14	44



Gambar 4.18. Persebaran klon SQL dalam dan antar kelas

Berdasarkan hasil Tabel 4.3. dan Gambar 4.18 menunjukkan klon SQL dapat ditemukan baik dalam satu kelas maupun antar kelas.

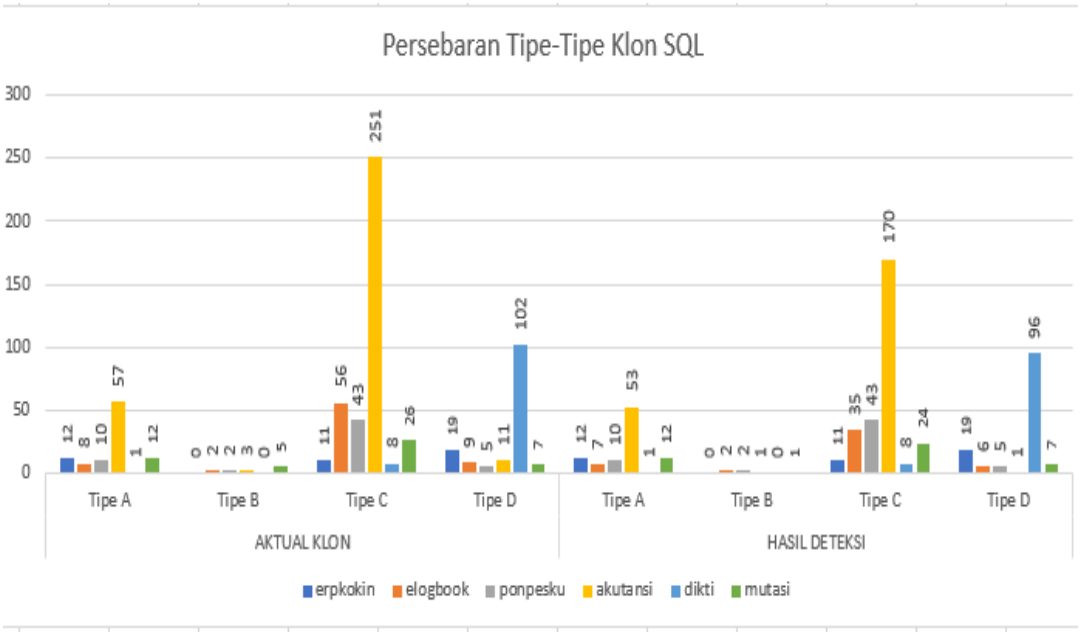
4.4.2.2.Persebaran Tipe-Tipe Klon SQL

Tujuan penelitian ini adalah mendeteksi klon SQL pada suatu aplikasi berarsitektur MVC. Untuk mendeteksi klon SQL tersebut, sebelumnya dilakukan pendefinisian tipe-tipe klon pada SQL karena masih belum ada penelitian yang membahas secara spesifik tipe-tipe klon pada SQL.

Dalam penelitian ini didefinisikan 4 tipe klon SQL yaitu tipe A, B, C dan D. Sehingga tujuan penelitian ini adalah mendeteksi apakah tipe-tipe tersebut ada pada aplikasi yang dijadikan dataset. Hasil deteksi dari tipe-tipe klon yang terdeteksi di dataset ditunjukkan pada Gambar 4.19. dan Tabel 4.4. berikut.

Tabel 4.4. Persebaran tipe-tipe klon

Dataset	Klon SQL label					Hasil deteksi sistem						
	A	B	C	D	total	A	B	C	D	Total (TP)	Kesalahan deteksi	Gagal deteksi
erpkokin	12	0	11	19	42	12	0	11	19	42	0	0
elogbook	8	2	56	9	75	7	2	35	6	50	0	25
ponpesku	10	2	43	5	60	10	2	43	5	60	0	0
akutansi	57	3	251	11	322	53	1	170	1	225	16	81
WS PD Dikti	1	0	8	102	111	1	0	8	96	105	6	0
Mutasi	12	5	26	7	50	12	1	24	7	44	3	3

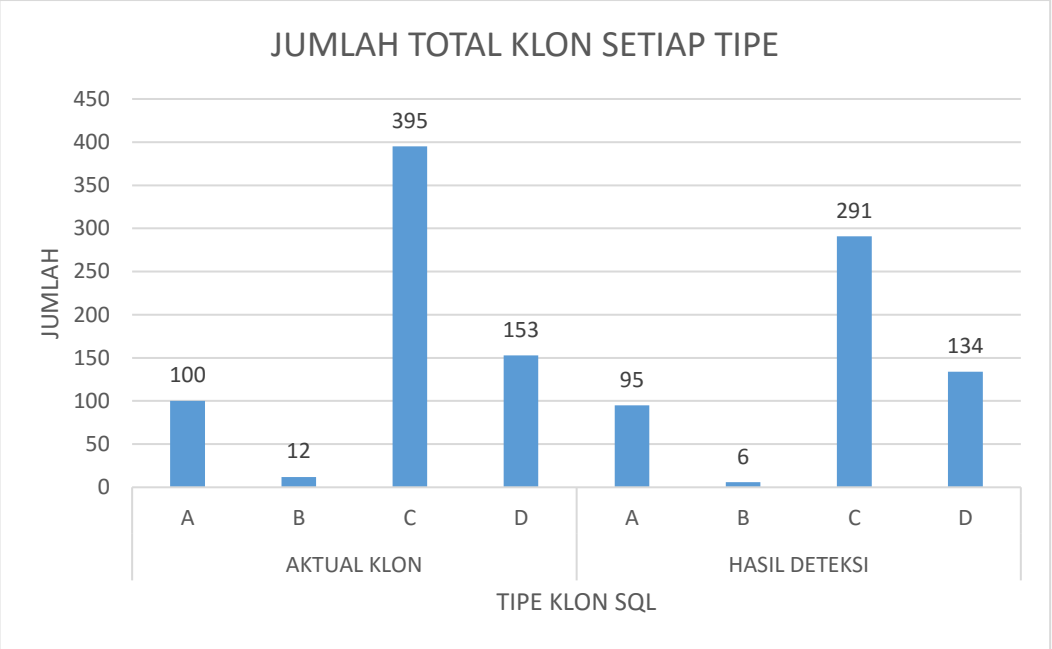


Gambar 4.19. Persebaran tipe-tipe klon SQL hasil deteksi

Tabel 4.4. menunjukkan hasil persebaran klon dari ujicoba yang dilakukan. Pada aplikasi Erpkokin dan Ponpesku tidak ada kesalahan dan kegagalan deteksi. Pada aplikasi Elogbook terjadi 25 gagal deteksi. Pada aplikasi Akutansi terdapat kasus 81 gagal deteksi dan 16 kesalahan deteksi. Kasus-kasus gagal dan kesalahan deteksi pada aplikasi Akutansi dan Elogbook karena terjadi kegagalan pada proses ekstraksi kueri. Pada aplikasi Web Service Pangkalan Data (PD) Dikti terdapat kasus 6 kesalahan deteksi. Sedangkan pada aplikasi Mutasi terdapat kasus 3 gagal deteksi dan 3 kesalahan deteksi.

Gambar 4.19. menunjukkan hasil persebaran klon dari ujicoba yang dilakukan. Dari gambar tersebut aplikasi yang paling banyak terdeteksi klon adalah apliaksi Akutansi dan Web Service Pangkalan Data (PD) Dikti.

Gambar 4.20. menunjukkan jumlah total klon setiap tipe pada seluruh aplikasi yang diuji. Tipe klon yang paling banyak ditemukan adalah klon tipe C. sedangkan tipe klon yang paling sedikit ditemukan adalah klon tipe B.



Gambar 4.20. Jumlah Total Klon Setiap Tipe

Contoh kode dan skrip SQL yang berhasil dideteksi sebagai klon SQL tipe A dapat ditunjukkan pada aplikasi Ponpesku pada kelas MKelas.

Tabel 4.5. SQL Klon tipe A dari raw SQL dan query builder

Kode Program	**Skrip SQL**

`//KELAS: MKelas`
`function get_kelas(){`
`  $stm="select * from kelas";`
`  return $this->db->query($stm)->result();`
`}`
	`select * from kelas`

`//KELAS: MKelas`
`function get_kelas_all(){`
`  return $this->db->get("kelas")->result();`
`}`
	`select * from kelas`

Tabel 4.5 menunjukkan klon SQL tipe A pada aplikasi Ponpesku. Dua skrip SQL tersebut ditulis dengan cara yang berbeda. Pada fungsi get_kelas() kueri ditulis dengan menggunakan raw SQL sedangkan pada fungsi get_kelas_all() kueri ditulis dengan menggunakan query builder dari kerangka kerja CodeIgniter.

Contoh kode dan skrip SQL yang berhasil dideteksi sebagai klon SQL tipe B dapat ditunjukkan pada Tabel 4.6 di aplikasi Ponpesku pada kelas M_kelas dan M_raport.

Tabel 4.6. SQL Klon Tipe B

Kode Program	Skrip SQL

//KELAS: MKelas
 	
function get_kelas_santri_detail($key){
. . . .
$stm="SELECT ks.*, u.nama,k.content FROM kelas_santri ks, ustadz u, kelas k WHERE ks.k_kelas=k.k_kelas AND ks.wali_kelas=u.k_ustadz AND ks.k_kelas_santri=?";
 	
$res1=$this->db->query($stm,array($key))->result();
. . . .
}
	SELECT ks.*, u.nama, k.content
	FROM kelas_santri ks, ustadz u, kelas k
	WHERE ks.k_kelas=k.k_kelas ANDks.wali_kelas=u.k_ustadz
	AND ks.k_kelas_santri=1

//KELAS: MKelas
 	
function generate_raport($data){
. . . .
$stm_data_kelas = "SELECT ks.*,u.nama ustadz,k.content FROM kelas_santri ks, ustadz u, kelas k WHERE ks.wali_kelas=u.k_ustadz AND ks.k_kelas=k.k_kelas AND ks.k_kelas_santri=?";
 	
$hasil_data_kelas = $this->db->query($stm_data_kelas,array($data['k_kelas_santri']))->result();
. . . .
}
	SELECT ks.*,u.nama ustadz,k.content
	FROM kelas_santri ks, ustadz u, kelas k
	WHERE ks.wali_kelas=u.k_ustadz AND
	ks.k_kelas=k.k_kelas AND
	ks.k_kelas_santri=1

Perbedaan dari dua skrip kueri diatas ada pada penamaan alias pada klusa SELECT yaitu:

SELECT ks.*, u.nama, k.content

Dengan

SELECT ks.*,u.nama ustadz,k.content

Selain perbedaan pada klausa SELECT, dua kueri diatas memiliki perbedaan urutan pada klausa filter.

Contoh kode dan skrip SQL yang berhasil dideteksi sebagai klon SQL tipe C dapat ditunjukkan pada Tabel 4.7 di aplikasi Elogbook pada kelas LoginMdl dan MasterMdl.

Tabel 4.7. SQL Klon Tipe C

Kode Program	Skrip SQL

//KELAS: LoginMdl
function login ($usr, $pwd){
. . .
$ss = $this->db->get_where('log_aqwers',
array('users' => $usr, 'passw' => $pwd));
. . .
}
	SELECT * FROM log_aqwers
	WHERE users = 1 AND passw
	= 1

//KELAS: MasterMdl
function cekUser ($id_pesankamar){
. . .
$nr = $this->db->from('log_aqwers')->where('nip',
$nip)->get()->num_rows();
. . .
}
	SELECT * FROM log_aqwers
	WHERE nip = 1

Perbedaan dari dua skrip kueri diatas ada pada penamaan alias pada kalusa filter yaitu:

```
WHERE users = 1 AND passw = 1
```

Dengan

```
WHERE nip = 1
```

Contoh kode dan skrip SQL yang berhasil dideteksi sebagai klon SQL tipe C yang memenuhi syarat minimal memiliki satu tabel yang sama dapat ditunjukkan pada Tabel 4.8. di aplikasi Akutansi.

Tabel 4.8. SQL Klon Tipe C dengan syarat minimal satu tabel

No	Kueri
Q1	SELECT SUM(RNC_VOLUME*RNC_HARGA) AS TOTAL FROM AKN_RINCIAN A JOIN AKN_SETREKENING B ON A.REK_ID = B.REK_ID JOIN AKN_ANGGARAN C ON A.ANG_ID = C.ANG_ID WHERE C.KEG_KODE = 1 AND RNC_TANGGAL BETWEEN 1 AND 1
Q2	SELECT SUM(RNC_VOLUME*RNC_HARGA) AS TOTAL FROM AKN_RINCIAN A JOIN AKN_SETREKENING B ON A.REK_ID = B.REK_ID WHERE ANG_ID = 1 AND RNC_TANGGAL BETWEEN 1 AND 1

Contoh kode dan skrip SQL yang berhasil dideteksi sebagai klon SQL tipe D dapat ditunjukkan pada Tabel 4.9 di aplikasi ERPKokin pada kelas Mpenjualan dan Mproduksi.

Tabel 4.9. SQL Klon Tipe D

Kode Program	Skrip SQL
//KELAS: Mpenjualan function insPesanan($pemesan, $jumlah,$produk, $tanggal,$idpesanan){ . . . . . $*statement*2 = 'select idharga_produk as id from harga_produk where tgl_akhir_berlaku = (select max(tgl_akhir_berlaku) from harga_produk where idproduk = ?) and idproduk = ?'; $maksi = $this->db->query($*statement*2, array($produk,$produk)); . . . . }	select idharga_produk as id from harga_produk where tgl_akhir_berlaku = (select max(tgl_akhir_berlaku) from harga_produk where idproduk = 1) and idproduk = 1
//KELAS: Mproduksi function insHarga($data){ . . . . $*statement*3 = 'select max(tgl_akhir_berlaku) as max from harga_produk where idproduk = ?'; $maxi = $this->db->query($*statement*3,$data['produk']); . . . .}	select max(tgl_akhir_berlaku) as max from harga_produk where idproduk = 1

Skrip SQL dari fungsi insHarga() di kelas Mproduksi merupakan klon dari subkueri skrip SQL fungsi Mpenjualan() di kelas Mpenjualan.

Selain klon tipe D dari subkueri, Klon SQL pada klausa UNION juga dapat dideteksi di aplikasi Erpkokin. Seperti ditunjukkan kueri Q1 dan kueri Q2 pada Tabel 4.10. berikut.

Tabel 4.10. SQL Klon Tipe D dengan klausa UNION

No	Kueri
Q1	SELECT masa_pajak_bulan, tahun_pajak, pajak_terutang_sekarang, pajak_terutang_sekarang FROM sptpd WHERE k_sptpd = 1 AND id_hotel = 1
Q2	SELECT COALESCE(masa_pajak_bulan, '00') masa_pajak_bulan, COALESCE(tahun_pajak, '0000') tahun_pajak, COALESCE(CONCAT(pajak_terutang_utang),'0') pajak_terutang_utang, COALESCE(pajak_terutang_utang,0) + COALESCE((pajak_terutang_utang*0.02),0) pajak_terutang_denda FROM pajak_utang WHERE k_sptpd = 1 AND id_hotel = 1 GROUP BY masa_pajak_bulan, tahun_pajak UNION ALL SELECT masa_pajak_bulan, tahun_pajak, pajak_terutang_sekarang, pajak_terutang_sekarang FROM sptpd WHERE k_sptpd = 1 AND id_hotel = 1

Temuan lain pada penelitian ini adalah kasus penggunaan fungsi formatter pada SQL. yang dimaksud fungsi formatter disini adalah fungsi yang dapat merubah bentuk hasil dari sebuah kolom yang diambil. Seperti YEAR(), DATE_FORMAT(), MONTH() dan sebagainya. Penggunaan fungsi-fungsi tersebut merubah hasil dari sebuah kolom sehingga penggunaan fungsi tersebut dianggap bukan klon dengan pemanggilan kolom. Contoh yang ditemukan pada aplikasi Ponpesku sebagai berikut:

```
SELECT thn_masuk, jenis_kelamin,COALESCE(COUNT(nis),0) jum
FROM santri s WHERE jenis_kelamin=1 AND is_aktif=1 GROUP BY thn_masuk,
jenis_kelamin
```

Dengan

```
SELECT YEAR(thn_masuk) as thn_masuk,
jenis_kelamin,COALESCE(COUNT(nis),0) jum FROM santri s WHERE
jenis_kelamin='L' GROUP BY YEAR(thn_masuk), jenis_kelamin
```

Kolom `thn_masuk` merupakan kolom dengan format DATE sehingga hasil dari kolom thn_masuk pada kueri pertama berbeda dengan `YEAR(thn_masuk)` pada kueri yang kedua. Contoh lain juga ditemukan pada aplikasi mutasi.

```
SELECT *, DATE(wkt_checkin) tgl_checkin, TIME(wkt_checkin) jam_checkin,
DATE(wkt_checkout) tgl_checkout, TIME(wkt_checkout) jam_checkout FROM
detail_menginap WHERE id_detail_menginap = 1 AND no_induk = 1 AND
id_kamar_hotel = 1
```

Pada contoh kueri diatas terdapat penggunaan fungsi yang berbeda pada kolom yang sama, yaitu `DATE(wkt_checkin)` dan `TIME(wkt_checkin).` Dua pemanggilan fungsi tersebut akan menghasilkan bentuk data yang berbeda sehingga dapat disimpulkan bahwa pemanggilan fungsi pada suatu kolom dianggap berbeda dengan kolom itu sendiri.

Berdasarkan hasi uji coba yang dilakukan terkait dengan informasi persebaran tipe-tipe klon yang berhasil dideteksi dalam suatu aplikasi, manfaat dan tindakan yang dapat diambil dalam bidang pemeliharaan aplikasi untuk setiap tipe klon adalah:

- **Klon SQL tipe A**

  Klon tipe A merupakan klon yang sama persis atau identik. Sehingga SQL yang dideteksi sebagai klon tipe A sebaiknya dihilangkan agar dapat menurunkan kompleksitas dan ukuran kode aplikasi.

- **Klon SQL tipe B**

  Pasangan klon tipe B memiliki SQL yang sama tetapi memiliki perbedaan alias dan urutan. SQL yang dideteksi sebagai klon tipe B sebaiknya dihilangkan agar dapat menurunkan kompleksitas dan ukuran kode aplikasi. Tetapi, perlu diperhatikan pada saat penghapusan SQL klon tipe B, karena pasangan SQL klon tipe B tidak sama persis atau tidak identik.

- **Klon SQL tipe C**

  Salah satu tindakan yang dapat dilakukan untuk pasangan klon SQL tipe C adalah pembuatan pustaka. SQL yang dideteksi sebagi klon tipe C tidak dapat dihilangkan atau dihapus seperti klon tipe A dan B karena klon tipe C adalah pasangan SQL yang memiliki kolom pada klausa SELECT yang sama tetapi memiliki hasil dan tujuan yang berbeda.

- **Klon SQL tipe D**

  Informasi SQL klon tipe D dapat digunakan sebagai bahan analisa skrip SQL yang sering menjadi subkueri atau bagian dari kueri lain. Dari analisa tersebut memungkinkan adanya duplikat subkueri yang dapat disederhanakan menjadi sebuah fungsi SQL.

4.4.2.3. Kesalahan dan Kegagalan Deteksi

Berikut ini akan dibahas terkait kegagalan dan kesalahan deteksi dari hasil uji coba yang dilakukan. Kegagalan dan kesalahan deteksi disebabkan beberapa hal yang akan dijelaskan sebagai berikut:

- **Kegagalan Ekstraksi Kueri**

Pada penelitian ini proses ekstraksi kueri dilakukan dengan pendekatan statis. Pendekatan statis adalah pendekatan tanpa menjalankan kode program. Selain itu, pada penelitian ini mengabaikan alur control program sehingga pada beberapa kasus akan terjadi gagal ekstraksi. Kasus gagal ekstraksi akan sangat

berdampak pada terjadinya kegagalan deteksi. Tabel 4.11. berikut merupakan rekapitulasi jumlah kueri yang gagal diekstraksi.

Tabel 4.11. Perbandingan jumlah aktual skrip SQL dengan hasil ekstraksi

Dataset	Aktual SQL skrip	Jumlah SQL skrip gagal diekstraksi
Erpkokin	89	0
Elogbook	74	7
Ponpesku	130	6
Akutansi	191	27
Web service PD Dikti	50	0
Mutasi	99	0

Kegagalan deteksi dan tingkat akurasi yang rendah pada aplikasi Elogbook dan Akutansi disebabkan adanya kueri yang tidak berhasil diekstraksi. Kegagalan ekstraksi kueri akan sangat berdampak pada hasil deteski klon SQL. Karena jika skrip SQL yang gagal diekstraksi merupakan pasangan klon dari skrip SQL yang lain, maka klon tidak dapat dideteksi.

Tetapi, tidak semua kasus gagal ekstraksi otomatis mengalami kasus gagal deteksi. Pada aplikasi Ponpesku terdapat beberapa kueri yang gagal diekstraksi. Tetapi kueri-kueri yang gagal diekstraksi tersebut tidak memiliki pasangan klon/tidak klon. Sehingga tidak mempengaruhi hasil deteksi klon.

Pada kasus aplikasi Elogbook dan Akutansi, kueri yang gagal diekstraksi memiliki pasangan klon sehingga terjadi kegagalan deteksi klon. Selain itu, jumlah pasangan klon juga sangat mempengaruhi hasil deteksi. Kasus yang ditemukan pada aplikasi Elogbook, terdapat 9 skrip SQL yang merupakan klon yaitu `SELECT * FROM log_masteraktivitas [..klausa filter..]`. Satu pasangan klon terdiri dari dua skrip SQL, sehingga ada 36 pasangan klon. Tetapi, dari 9 skrip SQL, ada 3 skrip SQL yang gagal diekstraksi sehingga hanya ada 15 pasangan klon yang dideteksi. Hal tersebut menunjukkan bahwa semakin besar skrip SQL yang klon dan pada anggota klon tersebut terjadi kegagalan ekstraksi, maka semakin besar kegagalan deteksi yang terjadi.

Kasus gagal ekstraksi terjadi pada alur kontrol. Contoh kasus kegagalan ekstraksi kueri pada suatu fungsi yang ada pada aplikasi Ponpesku ditunjukkan sebagai berikut:

```
function get_negara($k_negara='x'){
        $this->db->select('*');
        $this->db->from('m_negara');
        if($k_negara!='x') $this->db->where('k_negara', $k_negara);
        $result = $this->db->get();
        $result= $result->result();
        return $result;
}
```

Potongan kode diatas merupakan kode fungsi dari kelas M_Santri di aplikasi Ponpesku. Fungsi tersebut memiliki dua kueri / skrip SQL yaitu:

- `Q1 = SELECT * FROM m_negara`
- `Q2 = SELECT * FROM m_negara WHERE k_negara = 1`

Tetapi pada sistem yang terekstraksi hanya satu kueri yaitu kueri "**`SELECT * FROM m_negara WHERE k_negara = 1`**".

Kasus gagal ekstraksi lain ditemukan pada aplikasi Akutansi ditunjukkan sebagai berikut:

```
function realisasi($idn, $month, $year, $kelasposisi) {
    $lastday = cal_days_in_month(CAL_GREGORIAN, $month, $year);
    $sum = $kelasposisi == 'pendapatan' ? 'sum(bsr_kredit)' :
                          'sum(bsr_debet)';
    $sql = $this->db->query("select $sum as real from akn_setneraca a,
           akn_setrekening b, akn_bukubesar c where a.nrc_id = b.nrc_id and
           b.rek_id=c.rek_id and bsr_tanggal between '$year-01-01' and
           '$year-$month $lastday' and a.nrc_id = '$idn'");
    return $sql->row();
}
```

Pada fungsi realisasi tersebut terdapat dua skrip SQL yaitu:

- **Q1** = `select sum(bsr_kredit) as real from akn_setneraca a, akn_setrekening b, akn_bukubesar c where a.nrc_id = b.nrc_id and b.rek_id=c.rek_id and bsr_tanggal between 1 and 1 and a.nrc_id = 1`
- **Q2** = `select sum(bsr_debet) as real from akn_setneraca a, akn_setrekening b, akn_bukubesar c where a.nrc_id = b.nrc_id and b.rek_id=c.rek_id and bsr_tanggal between 1 and 1 and a.nrc_id = 1`

Tetapi pada sistem gagal mengekstraksi dengan benar kueri yang ada pada fungsi tersebut. Hasil ekstraksi sistem sebagai berikut:

- `select 1 as real from akn_setneraca a, akn_setrekening b, akn_bukubesar c where a.nrc_id = b.nrc_id and b.rek_id=c.rek_id and bsr_tanggal between 1 and 1 and a.nrc_id = 1`

Pada penelitian ini juga ditemukan kasus gagal ekstraksi kueri jika suatu kueri menjadi argumen dari klausa control *statement*. Kasus tersebut ditemukan di aplikasi Akutansi pada kelas piutanmdl.

```
function svPiutang($data) {
    if ($this->db->get_where('akn_piutang', array('ptg_id' =>
$data['ptg_id']) )->num_rows() > 0) {

    $sql = $this->db->where('ptg_id', $data['ptg_id'])->update('akn_piutang',
$data);
   } else {
     $sql = $this->db->insert('akn_piutang', $data);
   }
     return $sql;
}
```

Pada penelitian ini proses ekstraksi kueri dilakukan dengan pendekatan statis. Pendekatan statis adalah pendekatan tanpa menjalankan kode program. Selain itu, pada penelitian ini mengabaikan adanya persyaratan dari alur control sehingga pada beberapa kasus akan terjadi gagal deteksi, seperti pada contoh diatas.

Permasalahan lain pada proses ekstraksi kueri adalah pembentukan kueri yang dinamis diluar lapisan model yang ditunjukkan pada contoh berikut.

```
function ssp($sWhere, $sOrder, $sLimit, $jnsjab) {
. . . . .
    $query = $this->db->query("
        SELECT * FROM (
            SELECT *
            FROM log_masteraktivitas $jns
        ) A
        $sWhere
        $sOrder
        $sLimit
    ");
. . . .
}
```

Potongan kode diatas merupakan kode fungsi dari kelas MasterMdl di aplikasi Elogbook. Klausa WHERE, ORDER BY dan LIMIT dibentuk dan ditentukan diluar lapisan model. Klausa-klausa tersebut ditentukan oleh pemanggil fungsi pada lapisan model, sehingga pada saat ekstraksi kueri tidak dapat diekstraksi dengan tepat oleh sistem.

Enam aplikasi yang dijadikan objek uji coba memiliki karakteristik berbeda-beda. Aplikasi ERPkokin, Ponpesku, dan Mutasi adalah aplikasi yang sebagian besar kuerinya menggunakan bentuk *raw* SQL. Aplikasi Elogbook dan Akutansi adalah aplikasi yang sebagian besar kuerinya menggunakan bentuk *query builder*. Sedangkan kueri pada aplikasi WS PD Dikti sudah dalam bentuk SQL skrip. Jika dianalisa dari karakteristik aplikasi dikaitan dengan hasil deteksi, aplikasi yang sebagian besar menggunakan bentuk *query builder* memiliki hasil deteksi yang lebih rendah dibandingkan dengan *raw* SQL. Hal tersebut terjadi karena kueri yang dibangun dengan menggunakan *query builder* memiliki bentuk yang lebih dinamis dibandingkan dengan kueri yang dibentuk dengan *raw* SQL. Sehingga kasus gagal ekstraksi lebih banyak ditemukan pada kueri yang dibentuk dengan *query builder*. Contoh bentuk *query builder* yang gagal diekstraksi ditemukan di aplikasi Elogbook.

```
$query = $this->db->from('log_masteraktivitas')
         ->like('lower(bk_nama_kegiatan)', strtolower($word))
         ->order_by('bk_nama_kegiatan', 'asc');
$query->limit($limit, $start);
return $query->get()->result_array();
```

Kueri pada aplikasi Elogbook di atas menunjukkan salah satu bentuk *query builder* yang dinamis. Kueri tersebut dibentuk dengan kombinasi penugasan variabel dan pemanggilan fungsi kueri builder secara berantai (*Method Chaining*). Sistem gagal mengekstraksi baris ekspresi kode `$query->limit($limit, $start)` sehingga kueri yang berhasil diekstraksi sistem adalah:

```
$query = $this->db->from('log_masteraktivitas')
         ->like('lower(bk_nama_kegiatan)', strtolower($word))
         ->order_by('bk_nama_kegiatan', 'asc');
return $query->get()->result_array();
```

Kueri yang gagal terekstraksi dengan benar tersebut memiliki tujuh kueri yang merupakan klon sehingga kegagalan ekstraksi ini akan berdampak pada hasil akurasi, precision dan recall deteksi.

Dari beberapa kasus kegagalan ekstraksi yang telah dijelaskan maka dapat disimpulkan bahwa kegagalan ekstraksi kueri akan sangat berdampak pada hasil deteski klon SQL. Karena jika skrip SQL yang gagal diekstraksi merupakan pasangan klon dari skrip SQL yang lain, maka klon tidak dapat dideteksi.

- **Penggunaan Ekspresi pada Klausa SELECT**

Pendekatan yang diusulkan memiliki beberapa limitasi. Salah satunya adalah penggunaan ekspresi pada klausa SELECT. Pendekatan yang diusulkan masih belum dapat mendeteksi dua ekspresi yang ekuivalen dengan bentuk penulisan yang berbeda. Contoh kasus tersebut ditemukan di aplikasi mutase yang ditunjukkan pada Tabel 4.12 berikut:

Tabel 4.12. Contoh kasus penggunaan ekspresi klausa SELECT

skrip SQL 1	**skrip SQL 2**
`SELECT (SUM(dasar_pengenaan_utang)+SUM(pajak_terutang_utang)) as utang_total FROM pajak_utang WHERE k_sptpd = 1`	`SELECT (SUM(pajak_terutang_utang)+SUM(dasar_pengenaan_utang)) as utang_total FROM pajak_utang WHERE k_sptpd = 1`

Dua ekspresi tersebut tidak dianggap klon, karena memilki urutan yang berbeda, selain itu ekspresi pada SELECT *statement* pada contoh tersebut akan dianggap sebagai satu kolom. Sehingga dua ekspresi tersebut dianggap tidak sama. Terkait dengan hal tersebut perlu dilakukan pengkajian lebih mendalam terhadap penggunaan ekspresi dan ekuivalensi ekspresi pada skrip SQL.

- **Penggunaan Subkueri pada Klausa FROM**

Selain penggunaan ekspresi, pendekatan yang diusulkan memiliki limitasi pada perbandingan tabel klausa FROM yang dibentuk dari subkueri. Pendekatan yang diusulkan mengkomparasikan tabel yang dibentuk dari subkueri sebagai satu tabel utuh tanpa melihat lebih dalam pada struktur subkueri tersebut. Sehingga jika ada dua tabel yang dibentuk dari subkueri dikomparasikan, hanya tabel yang dibentuk dari subkueri yang sama persis yang dianggap sebagai tabel yang sama.

Tetapi jika salah satu subkueri pembentuk tabel memiliki struktur yang berbeda, misalkan tipe B, maka dua tabel yang dibentuk dari subkueri tersebut akan dianggap berbeda. Penjelasan lebih detail ditunjukkan pada contoh kueri berikut ini:

```
SELECT subA.colA1,sub.colB1
FROM
(SELECT colA1,colA2 FROM tabelA) as tabA,
(SELECT colB1,colB2 FROM tabelB) as tabB,
WHERE tabA.colA1 = tabB.colB1
```

```
SELECT subA.colA1,sub.colB1
FROM
(SELECT colA1,colA2 FROM tabelA) as tabA,
(SELECT colB1,colB2 FROM tabelB) as tabB,
WHERE tabA.colA1 = tabB.colB1
```

Dari contoh di atas, kedua kueri tersebut dianggap klon tipe A karena kedua kueri tersebut memiliki tulisan yang sama persis. Pada klausa FROM, dua kueri tersebut dideteksi memiliki tabel yang sama, walaupun tabel dibentuk dari subkueri. Tetapi pada kasus yang ditemukan di aplikasi mutase ditunjukkan pada Tabel 4.13 berikut:

Tabel 4.13. Contoh kasus penggunaan subkueri pada klausa FROM

skrip SQL 1	**skrip SQL 2**
SELECT gol.id_gol_kamar, gol.harga, kam.jml_kamar FROM (SELECT harga, id_gol_kamar FROM m_golongan_kamar WHERE id_hotel = 1) gol LEFT JOIN (SELECT COUNT(id_kamar_hotel)jml_kamar, id_gol_kamar FROM kamar_hotel GROUP BY id_gol_kamar)kam ON gol.id_gol_kamar = kam.id_gol_kamar	SELECT gol.id_gol_kamar, gol.harga, kam.jml_kamar FROM (SELECT id_gol_kamar, harga FROM m_golongan_kamar WHERE id_hotel = 1) gol LEFT JOIN (SELECTid_gol_kamar, COUNT(id_kamar_hotel)jml_kamar FROM kamar_hotel GROUP BY id_gol_kamar)kam ON gol.id_gol_kamar =kam.id_gol_kamar

Pada Tabel 4.13, kedua kueri tidak dianggap sebagai klon pada saat proses deteksi. Karena dianggap memiliki tabel yang berbeda. Kedua kueri gagal dideteksi sebagai klon karena perbedaan pada dua subkueri pembentuk tabel. Kedua subkueri pembentuk tabel tersebut sebenarnya merupakan kueri yang sama, hanya berbeda pada urutan kolom pada klausa SELECT (klon tipe B). Seharusnya dua subkueri pembentuk tabel tersebut tetap dianggap sebagai tabel yang sama sehingga

kasus seperti ini tetap dideteksi sebagai klon B. Oleh karena itu kasus seperti ini perlu dikaji lebih mendalam.

- **Penggunaan Alias Posisi pada ORDER BY dan GROUP BY**

Kesalahan deteksi ditemukan pada dataset mutasi. Kesalahan deteksi tersebut terjadi karena pendekatan yang diusulkan tidak dapat mendeteksi penggunaan alias dengan posisi pada klausa GROUP BY dan ORDER BY. Pada klausa GROUP BY dan ORDER BY memungkinkan penggunaan nomor posisi pada klausa SELECT. Contoh kasus yang ditemukan ditunjukkan pada Tabel 4.14. sebagai berikut:

Tabel 4.14. Contoh kasus Penggunaan Alias Posisi pada ORDER BY

SQL skrip 1	**SQL skrip 2**
`select content kamar, harga from m_golongan_kamar order by 2`	`select content kamar, harga from m_golongan_kamar order by harga`

Klausa order by 2 dan order by harga adalah ekuivalen. Nomor 2 adalah merujuk pada kolom yang sama yaitu kolom harga. Oleh sebab itu seharusnya masuk dalam klon tipe B. sedangkan sistem mendeteksi sebagai klon tipe C karena dianggap memiliki klausa filter yang berbeda. Temuan ini dapat dijadikan acuan untuk memperbaiki definisi yang ada.

Tabel 4.15. Contoh kasus Penggunaan Alias Posisi pada GROUP BY

SQL skrip 1	**SQL skrip 2**
`SELECT h.nama, COUNT(k.id_kamar_hotel)jml_kamar FROM kamar_hotel k, m_hotel h WHERE k.id_hotel=h.id_hotel GROUP BY 1`	`SELECT h.nama, COUNT(k.id_kamar_hotel)jml_kamar FROM kamar_hotel k, m_hotel h WHERE k.id_hotel=h.id_hotel GROUP BY h.nama`

Temuan lain penggunaan alias posisi klausa select juga ditemukan pada klausa GROUP BY. Pada Tabel 4.15. klausa GROUP BY 1 dengan klausa GROUP BY h.nama adalah ekuivalen.

- **Urutan LEFT JOIN dan RIGHT JOIN**

Salah satu poin pada definisi klon tipe B adalah memiliki urutan yang berbeda. Syarat urutan yang berbeda ini berlaku untuk klausa SELECT, FROM dan filter. Tabel 4.16. berikut adalah contoh klon tipe B terkait dengan penggunaan klausa JOIN ditemukan pada aplikasi Elogbook.

Tabel 4.16. Contoh kasus terkait urutan pada JOIN

SQL skrip 1	SQL skrip 2
`SELECT * FROM skp_jabatan_penilai JOIN skp_pns ON jb_kode_parent = kode_jabatan WHERE jb_kode_child = 1`	`SELECT * FROM skp_pns JOIN skp_jabatan_penilai ON kode_jabatan = jb_kode_parent WHERE jb_kode_child = 1`

Pasangan skrip SQL diatas merupakan klon tipe B karena adanya perbedaan urutan tabel pada klausa FROM, yaitu `FROM skp_jabatan_penilai JOIN skp_pns` dengan `FROM skp_pns JOIN skp_jabatan_penilai`.

Tetapi, pada temuan lain, perbedaan urutan tabel pada klausa LEFT JOIN dan RIGHT JOIN akan memberikan hasil yang berbeda dibandingkan dengan perbedaan urutan tabel pada klausa JOIN seperti pada contoh sebelumnya.

Kasus perbedaan urutan tabel pada klausa LEFT JOIN ditemukan di aplikasi Mutasi ditunjukkan pada Tabel 4.17 berikut:

Tabel 4.17. Contoh kasus terkait urutan LEFT JOIN dan RIGHT JOIN

SQL skrip 1	SQL skrip 2

`SELECT sp.no_sptpd, hot.nama, hot.npwpd, hot.alamat`
`FROM sptpd sp`
`LEFT JOIN m_hotel hot ON sp.id_hotel = hot.id_hotel`
`ORDER BY tanggal_kirim DESC`
	`SELECT sp.no_sptpd, hot.nama, hot.npwpd, hot.alamat`
	`FROM m_hotel hot`
	`LEFT JOIN sptpd sp ON sp.id_hotel = hot.id_hotel`
	`ORDER BY tanggal_kirim DESC`

Pada kasus di atas, sistem mendeteksi sebagai klon tipe B. seharusnya dua skrip SQL tersebut digolongkan kedalam klon tipe C karena kedua skrip SQL tersebut memiliki hasil yang berbeda. Urutan tabel pada klausa LEFT JOIN dan RIGHT JOIN berpengaruh pada hasil data yang ditampilkan.

Temuan terkait urutan tabel pada LEFT JOIN dan RIGHT JOIN dapat dijadikan masukan untuk perbaikan definisi tipe-tipe klon selanjutnya.

- **Ekuivalensi penggunaan LEFT JOIN dan RIGHT JOIN**

Dari hasil uji coba ditemukan kasus kueri yang menggunakan klausa LEFT JOIN dan RIGHT JOIN tetapi memiliki tujuan dan hasil yang ekuivalen. Kasus tersebut ditemukan di aplikasi Mutasi yang ditunjukkan pada Tabel 4.18 berikut:

Tabel 4.18. Kasus ekuivalensi penggunaan LEFT JOIN dan RIGHT JOIN

SQL skrip 1	**SQL skrip 2**

`SELECT h.nama, COUNT(s.jns_service)`
`FROM m_hotel h`
`LEFT JOIN m_service_hotel s`
`ON h.id_hotel = s.id_hotel`
`GROUP BY h.nama`
	`SELECT h.nama, COUNT(s.jns_service)`
	`FROM m_service_hotel s`
	`RIGHT JOIN m_hotel h`
	`ON h.id_hotel = s.id_hotel`
	`GROUP BY h.nama`

Dua skrip SQL tersebut merupakan kueri yang equivalen walaupun memiliki perbedaan klausa join. Hasil deteksi sistem menunjukkan bahwa pasangan skrip SQL tersebut merupakan klon tipe C karena memiliki klausa join yang berbeda. Dari sisi definisi, pasangan skrip SQL tersebut juga merupakan klon tipe C. Sehingga dari kasus tersebut dapat dijadikan masukan untuk perbaikan definisi untuk tipe-tipe klon SQL.

- **Perbedaan urutan pada UNION**

Dari hasil uji coba ditemukan kasus UNION yang ekuivalen tetapi memiliki urutan yang berbeda gagal dideteksi oleh sistem. Kasus tersebut ditemukan di aplikasi Mutasi yang ditunjukkan pada Tabel 4.19. berikut:

Tabel 4.19. Kasus perbedaan urutan pada UNION

SQL skrip 1	**SQL skrip 2**

`SELECT id_hotel, tanggal_terima,`
`file_name`
`FROM skpd WHERE id_hotel = $id_hotel`
`UNION`
`SELECT id_hotel,`
`tanggal_terima_hotel, file_name`
`FROM sptpd WHERE id_hotel = $id_hotel`
`UNION`
`SELECT id_hotel, tanggal_terima,`
`file_name`
`FROM sspd WHERE id_hotel = $id_hotel`
	`SELECT id_hotel, tanggal_terima,`
	`file_name`
	`FROM skpd WHERE id_hotel = $id_hotel`
	`UNION`
	`SELECT id_hotel, tanggal_terima,`
	`file_name`
	`FROM sspd WHERE id_hotel = $id_hotel`
	`UNION`
	`SELECT id_hotel,`
	`tanggal_terima_hotel, file_name`
	`FROM sptpd WHERE id_hotel = $id_hotel`

Dua skrip SQL diatas masuk dalam kriteria klon tipe B karena memiliki perbedaan urutan. Tetapi, sistem tidak dapat mendeteksi pasangan skrip tersebut sebagai klon karena perbedaan urutan pada UNION *statement*. Perbedaan urutan pada UNION *statement* ini dapat dijadikan masukan untuk perbaikan definisi tipe-tipe klon SQL.

- **Penggunaan Prefiks Tabel pada klausa FROM**

Dari hasil uji coba ditemukan kasus terkait dengan penggunaan prefiks tabel pada klausa FROM yang berpengaruh pada kesalahan deteksi. Kasus tersebut ditemukan di aplikasi Web Service PD Dikti yang ditunjukkan pada Tabel 4.20. berikut:

Tabel 4.20. Contoh kasus penggunaan prefiks tabel pada klausa FROM

SQL skrip 1	**SQL skrip 2**
`SELECT r.id_sdm, MAX(r.id_jabfung) AS max_jabfung FROM rwy_fungsional r WHERE r.soft_delete = 0 GROUP BY r.id_sdm`	`SELECT r.id_sdm, MAX(r.id_jabfung) AS max_jabfung FROM dbo.rwy_fungsional r WHERE r.soft_delete = 0 GROUP BY r.id_sdm`

Tabel `rwy_fungsional` dan `dbo.rwy_fungsional` merupakan tabel yang sama dan dari skema database yang sama. `dbo` adalah skema default dari databasenya sehingga penggunaan prefix dbo pada skema dbo seharusnya diabaikan. Dua skrip tersebut seharusnya klon tipe A, tetapi sistem mendeteksi sebagai klon tipe B karena perbedaan prefix `dbo.`. kesalahan deteksi tersebut dikarenakan pendekatan yang diusulkan tidak mengakomodasi struktur database dan tabel. Seharusnya prefiks pada tabel juga dipertimbangkan karena prefiks pada tabel di klausa FROM menspesifikasikan informasi struktur basis data.

Selain itu, prefiks pada tabel dapat mensepesifikasikan dua tabel yang memiliki nama yang sama tetapi dari basis data skema yang berbeda. Tabel 4.21. berikut merupakan contoh penggunaan prefiks tabel yang berbeda pada klausa FROM:

Tabel 4.21. Contoh kasus perbedaan prefiks pada tabel

SQL skrip 1	SQL skrip 2
`SELECT col1, col2, col3 FROM mydb.tableA WHERE col3>10 ORDER BY col3`	`SELECT col1, col2, col3 FROM backup.tableA WHERE col3>10 ORDER BY col3`

Pada contoh kasus diatas `mydb.tableA` dan `backup.tableA` merupakan tabel yang berbeda. Hal tersebut ditunjukkan dengan prefiks tabel yang berbeda. Penggunaan prefiks pada tabel memungkinkan digunakan untuk menspesifikkan tabel yang digunakan pada klausa FROM.

Berdasarkan definisi dan pendekatan deteksi pada penelitian ini, dua skrip SQL tersebut masuk dalam kriteria klon tipe B. Seharusnya, dua skrip SQL tersebut

adalah bukan klon karena dari tabel yang berbeda. Sehingga contoh kasus penggunaan prefiks tabel pada klausa FROM dapat dijadikan masukkan untuk perbaikan definisi klon SQL tipe B.

Untuk mengakomodasi prefix pada tabel diperlukan informasi nama basis data dan struktur basis data yang digunakan. Penambahan infromasi tersebut pada proses deteksi akan menambah keakuratan hasil deteksi.

- **Ekuivalensi Select all**

Pada definisi dan pendekatan yang diusulkan terdapat limitasi terkait penggunaan SELECT * dan SELECT dengan menyebutkan seluruh kolom pada tabel. Misalnya pada tabel A memiliki struktur kolom: column1, column2, column3. Maka contoh ekuivalensi klausa select all ditunjukkan pada Tabel 4.22. berikut:

Tabel 4.22. Contoh kasus ekuivalensi SELECT all

SQL skrip 1	**SQL skrip 2**
`SELECT col1, col2, col3` `FROM tableA`	`SELECT *` `FROM tableA`

Pada contoh kasus diatas, dua skrip SQL tersebut ekuivalen. Dari kasus tersebut dapat dijadikan masukkan untuk perbaikan definisi klon SQL. Selain itu, untuk dapat mendeteksi ekuivalensi pada kasus select all, diperlukan penambahan infromasi struktur basis data pada proses deteksi.

BAB 5
PENUTUP

Pada bab ini dijelaskan mengenai kesimpulan akhir yang didapat setelah melakukan implementasi pendekatan usulan dan serangkaian ujicoba pada bab sebelumnya.

5.1. Kesimpulan

Kesimpulan yang dapat diambil dalam penelitian ini antara lain adalah sebagai berikut.

1. Penelitian ini mendefinisikan klon SQL dan tipe-tipe klon SQL. Tipe klon SQL yang didefinisikan adalah A, B, C, dan D. Tipe A didefinisikan sebagai sepasang skrip SQL yang memiliki tulisan yang sama persis. Tipe B didefinisikan sebagai sepasang skrip SQL memiliki klausa SELECT, klausa FROM, dan klausa kondisi yang sama, tetapi memiliki urutan dan penamaan (alias) yang berbeda di kolom dan tabel pada klausa SELECT, klausa FROM, dan klausa kondisi. Tipe C didefinisikan sebagai sepasang skrip SQL memiliki tiga syarat utama yaitu: memiliki klausa SELECT yang sama (tanpa memperhatikan perbedaan nama dan urutan), Pada klausa FROM memiliki tabel yang sama tetapi menggunakan cara join yang berbeda, atau minimal memiliki satu tabel yang sama. Pada klausa kondisi memiliki kondisi yang berbeda. Tipe D didefinisikan sebagai sepasang SQL yang termasuk klon tipe A, B, atau C, tetapi salah satu atau kedua kueri tersebut merupakan bagian dari sebuah kueri lain.
2. Pendekatan yang diusulkan untuk mendeteksi klon SQL sesuai dengan definisi dan tipe-tipe klon SQL memiliki hasil yang signifikan. Dari semua dataset yang ada, nilai rata-rata akurasi pendekatan yang diusulkan sebesar 86,52%. Sedangkan untuk nilai precision dan recall berturut-turut sebesar 94,85% dan 84,34%.
3. Definisi klon SQL dan tipe-tipe klon SQL yang didefinisikan pada penelitian ini terbukti ada pada aplikasi berbasis MVC. Hal tersebut dibuktikan dengan

hasil deteksi yang mendeteksi semua tipe-tipe klon yang telah didefinisikan yaitu tipe A, B, C dan D.

4. Tipe klon yang paling banyak ditemukan adalah klon tipe C. sedangkan tipe klon yang paling sedikit ditemukan adalah klon tipe B.

5.1. Saran

Saran yang dapat diberikan untuk pengembangan lebih lanjut penelitian ini adalah sebagai berikut.

1. Kegagalan ekstraksi kueri dapat berdampak pada hasil deteksi klon SQL. Sehingga perbaikan pada proses ekstraksi kueri diharapkan dapat memperbaiki hasil deteksi. Misalnya mengakomodasi ekstraksi dengan pendekatan dinamis, dan memperhatikan alur kontrol.
2. Selain *raw* SQL dan *query builder*, bentuk penulisan kueri dengan menggunakan ORM pada lapisan model dapat diperhitungkan sebagai bentuk kueri yang dapat diakomodasi pada proses ekstraksi kueri.
3. Perlu dikaji lebih mendalam terkait kegagalan dan kesalahan deteksi sehingga dapat menjadi acuan perbaikan definisi tipe-tipe klon SQL atau pendefinisian tipe baru klon SQL. Serta menjadi acuan perbaikan pendekatan yang diusulkan.
4. Penambahan informasi struktur database aplikasi dapat dijadikan bahan pertimbangan sebagai tambahan inputan sistem sehingga dapat mengatasi masalah penggunaan prefiks tabel pada klausa FROM dan ekuivalensi select all. Mengingat pada penelitian ini hanya berbasis skrip SQL.

DAFTAR PUSTAKA

Akbar, F. A. (2014). *RANCANG BANGUN SISTEM ERP (ENTERPRISE RESOURCE PLANNING) UNTUK PONDOK PESANTREN (Studi Kasus: Pondok Pesantren Salafiyah Syafi'iyah Nurul Huda, Mergosono-Malang)*. Universitas Brawijaya.

Akbarnejad, J. (2010). SQL QueRIE Recommendations : a query fragment-based approach, *3*(2).

Anderson, D., & Hills, M. (n.d.). Query Construction Patterns in PHP.

Arvin, T. (2014). Comparison of different SQL implementations. Retrieved April 6, 2017, from http://troels.arvin.dk/db/rdbms/

Bansal, G., & Tekchandani, R. (2014). Selecting a set of appropriate metrics for detecting code clones. *2014 7th International Conference on Contemporary Computing, IC3 2014*, 484–488. https://doi.org/10.1109/IC3.2014.6897221

Beaulieu, A. (2009). *Learning SQL, 2nd Edition*. O'Reilly Media.

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., & Merlo, E. (2007). Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, *33*(9), 577–591. https://doi.org/10.1109/TSE.2007.70725

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *PATTERN-ORIENTED SOFTWARE ARCHITECTURE* (4th ed.). West Sussex: John Wiley & Sons, Ltd.

Chatterji, D., Carver, J. C., & Kraft, N. A. (2012). Claims and beliefs about code clones: Do we agree as a community? A survey. *2012 6th International Workshop on Software Clones, IWSC 2012 - Proceedings*, 15–21. https://doi.org/10.1109/IWSC.2012.6227860

Cheung, W. T., Ryu, S., & Kim, S. (2016). Development nature matters: An empirical study of code clones in JavaScript applications. *Empirical Software Engineering*, *21*(2), 517–564. https://doi.org/10.1007/s10664-015-9368-6

Cordy, J. R., & Dean, T. R. (2004). Practical Language-Independent Detection of Near-Miss Clones.

Cordy, J. R., & Roy, C. K. (2011). The NiCad clone detector. *IEEE International Conference on Program Comprehension*, (Figure 3), 219–220. https://doi.org/10.1109/ICPC.2011.26

Das, D., Sharma, U., & Bhattacharyya, D. (2010). An Approach to Detection of SQL Injection Attack Based on Dynamic Query Matching. *International Journal of Computer …*, *1*(25), 28–34. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.7553&rep=rep1&type=pdf

dobarkod. (2017). django-queryinspect. Retrieved February 12, 2017, from

https://github.com/dobarkod/django-queryinspect

Eirinaki, M., Abraham, S., Polyzotis, N., & Shaikh, N. (2014). QueRIE : Collaborative Database Exploration, *26*(7), 1778–1790.

Hotta, K., Yang, J., Higo, Y., & Kusumoto, S. (2014). How Accurate Is Coarse-grained Clone Detection?: Comparision with Fine-grained Detectors. In *Proceedings of the Eighth InternationalWorkshop on Software Clones (IWSC 2014)* (Vol. 63). Electronic Communications of the EASST. https://doi.org/http://dx.doi.org/10.14279/tuj.eceasst.63.922

Islam, M. R., Islam, M. R., Islam, M. M., & Halim, T. (2011). A study of code cloning in server pages of web applications developed using classic ASP.NET and ASP.NET MVC framework. *14th International Conference on Computer and Information Technology, ICCIT 2011*, (Iccit), 497–502. https://doi.org/10.1109/ICCITechn.2011.6164840

Jiang, L., Misherghi, G., & Su, Z. (2007). DECKARD : Scalable and Accurate Tree-based Detection of Code Clones ∗, (520320).

Kamiya, T., Kusumoto, S., & Inoue, K. (2002). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, *28*(7), 654–670. https://doi.org/10.1109/TSE.2002.1019480

Koschke, R., Falke, R., & Frenzel, P. (2006). Clone detection using abstract syntax suffix trees. *Proceedings - Working Conference on Reverse Engineering, WCRE*. https://doi.org/10.1109/WCRE.2006.18

Leff, A., & Rayfield, J. T. (2001). Web-Application Development Using the Model/View/Controller Design Pattern. In *Enterprise Distributed Object Computing Conference* (pp. 118–127).

Mazinanian, D., Tsantalis, N., & Mesbah, A. (2014). Discovering Refactoring Opportunities in Cascading Style Sheets. *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 496–506. https://doi.org/10.1145/2635868.2635879

Morales-Chaparro, R., Linaje, M., Preciado, J. C., & Sánchez-Figueroa, F. (2007). MVC web design patterns and rich internet applications. In *Proceedings of the Jornadas de Ingeniería del Software y Bases de Datos* (pp. 39–46).

Moreau, E., Yvon, F., & Cappé, O. (2008). Robust similarity measures for named entities matching. *Computational Linguistics*, *1*(August), 593–600. https://doi.org/10.3115/1599081.1599156

Muhammad, T., Zibran, M. F., Yamamoto, Y., & Roy, C. K. (2013). Near-miss clone patterns in web applications: An empirical study with industrial systems. *Canadian Conference on Electrical and Computer Engineering*, 1–6. https://doi.org/10.1109/CCECE.2013.6567821

MySQL. (2017). MySQL Standards Compliance. Retrieved April 6, 2017, from

https://dev.mysql.com/doc/refman/5.6/en/compatibility.html

Oracle. (2017). SQL Standards. Retrieved April 6, 2017, from http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/intro002.htm

Popov, N. (2017). PHP-Parser. Retrieved March 17, 2017, from https://github.com/nikic/PHP-Parser

Priyambadha, B., & Rochimah, S. (2014). Case study on semantic clone detection based on code behavior. *Proceedings of 2014 International Conference on Data and Software Engineering, ICODSE 2014*. https://doi.org/10.1109/ICODSE.2014.7062689

Rajapakse, D. C., & Jarzabek, S. (2005). An investigation of cloning in web applications. *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web - WWW '05*, 924–935. https://doi.org/10.1145/1062745.1062800

Rattan, D., Bhatia, R., & Singh, M. (2013). *Software clone detection: A systematic review*. *Information and Software Technology* (Vol. 55). Elsevier B.V. https://doi.org/10.1016/j.infsof.2013.01.008

Rattan, D., & Kaur, J. (2016). Systematic Mapping Study of Metrics based Clone Detection Techniques.

Sheneamer, A., & Kalita, J. (2016). Code clone detection using coarse and fine-grained hybrid approaches. *2015 IEEE 7th International Conference on Intelligent Computing and Information Systems, ICICIS 2015*, 472–480. https://doi.org/10.1109/IntelCIS.2015.7397263

St??rrle, H. (2013). Towards clone detection in UML domain models. *Software and Systems Modeling*, *12*(2), 307–329. https://doi.org/10.1007/s10270-011-0217-9

Stephan, M. (2014). Model Clone Detector Evaluation Using Mutation Analysis. *2014 IEEE International Conference on Software Maintenance and Evolution*, 633–638. https://doi.org/10.1109/ICSME.2014.113

Swanhart, J. (2017). PHP-SQL-Parser. Retrieved March 17, 2017, from https://github.com/greenlion/PHP-SQL-Parser

Tuya, J., Suárez-cabal, M. J., & Riva, C. De. (2006). SQLMutation : A tool to generate mutants of SQL database queries.

Yudoyo, G. (2015). *Analisis dan Rancang Bangun Aplikasi Enterprise Resource Planning pada PT.Koki Indocan*. Universitas Brawijaya.

Zulkarnaein. (2014). *Rancang Bangun Sistem Informasi Pajak Perhotelan Online (Studi Kasus di DISPENDA Malang)*. Universitas Brawijaya.

(halaman ini sengaja dikosongkan)

BIOGRAFI PENULIS

Penulis dilahirkan di Sidoarjo pada tanggal 17 Maret 1992, yang merupakan anak kedua dari dua bersaudara. Penulis telah menempuh pendidikan dasar di SDN Jemirahan-Jabon, SMP Negeri 1 Porong, dan SMA Negeri 1 Sidoarjo. Pada tahun 2010 penulis melanjutkan pendidikan ke jenjang pendidikan tinggi di Teknik Informatika Universitas Brawijaya dan lulus tahun 2014. Kemudian, tahun 2015 penulis melanjutkan studi S2 di Institut Teknologi Sepuluh Nopemeber di Program Studi Teknik Informatika dan lulus sebagai Magister Komputer pada tahun 2017. Penulis mengambil bidang minat Rekayasa Perangkat Lunak baik di jenjang S1 maupun S2. Selain itu, penulis pernah menjadi pengajar di PIKTI ITS selama menempuh perkuliahan S2 di ITS. Untuk korespondensi, penulis dapat dihubungi melalui email ewezali@gmail.com.

(halaman ini sengaja dikosongkan)